

---

***Cardiff* LiquidOffice™ eForm Management System**

# LiquidOffice™ Custom Scripting Guide

*Version 2.1*

---

---

## Notice

This software program is a proprietary product of Cardiff Software, Inc. and is protected by copyright laws and international treaty. Use of this software is subject to acceptance of the Cardiff Software End User License Agreement included in this software package.

Information in this manual is subject to change without notice and does not represent a commitment on the part of Cardiff Software, Inc. The software described in this document is furnished under a license agreement which states the terms for use of this product. The software may be used or copied only in accordance with the terms of that agreement. No part of this publication may be reproduced, transmitted, stored in a retrieval system, or translated into another language without the written permission of Cardiff Software, Inc. This manual utilizes fictitious names for purposes of demonstration; references to actual persons, companies, or organizations is strictly coincidental.

## Trademarks and Copyright

Copyright © 2002 Cardiff Software, Inc. All Rights Reserved. The Cardiff logo, Cardiff, and LiquidOffice are trademarks or registered trademarks of Cardiff Software, Inc. The Adobe logo, Adobe and Acrobat are trademarks or registered trademarks of Adobe Systems Incorporated. This product includes Adobe PDF. Microsoft, Windows NT and Windows are trademarks or registered trademarks of Microsoft Corporation. Pentium is a registered trademark of Intel Corporation. Sun, Sun Microsystems, Solaris, Java, and JavaScript are trademarks or registered trademarks of Sun Microsystems, Inc. in the United States and other countries. All SPARC trademarks are used under license and are trademarks or registered trademarks of SPARC International, Inc. in the United States and other countries. Products bearing SPARC trademarks are based upon an architecture developed by Sun Microsystems, Inc. Oracle and Oracle8i are trademarks or registered trademarks of Oracle Corporation. Xerox and Document Centre are registered trademarks of Xerox Corporation. FormBridge and TEXCEL are registered trademarks of Texcel Systems, Inc. Portions of the LiquidOffice Form Designer product were created using LeadTools ©1991-2002, Lead Technologies, Inc. All Rights Reserved. The LZW technology is included with LiquidOffice Form Designer and is provided by Unisys and is licensed under U.S. Patent No. 4,558,302 and foreign counterparts. Portions of the LiquidOffice Form Designer program use the Sentry Spelling-Checker Engine. Copyright © 2002 Wintertree Software Inc. Portions of the LiquidOffice Form Designer program use technologies licensed from Inner Media, Rogue Wave, Sub Systems, and Compuware. Corporation. Copyright (c) 2002. Portions of the LiquidOffice Form Server program use technologies licensed under the Mozilla Public License Version 1.1. Source code can be found on [www.mozilla.org](http://www.mozilla.org). The LiquidOffice Form Server product includes software developed by the Apache Software Foundation (<http://www.apache.org/>). Copyright (c) 1999-2002 The Apache Software Foundation. The LiquidOffice Form Server product includes software developed by the JDOM Project (<http://www.jdom.org/>), copyright (c) 2000-2002 Brett McLaughlin & Jason Hunter. The Liquidoffice Form Server product includes software developed by the Jaxen Project (<http://jaxen.org>), copyright (c) 2000-2002 Bob McWhirter & James Strachan. All rights reserved.

## Patent Information

Covered by U.S. Patents 4,893,333; 5,24,7591; 5,555101 and 5,943137. Additional patents pending, including 60/261743.

Cardiff Software Incorporated, Vista, CA 92083  
[www.cardiff.com](http://www.cardiff.com)

## Document Number and Revision

LiquidOffice Custom Scripting Guide 100-00047 Rev A

---

---

# CONTENTS

<b>LiquidOffice Client-Side JavaScript Event Model.....</b>	<b>5</b>
Form Level Events .....	5
Field Level Events .....	6
<b>LiquidOffice Client-Side JavaScript Object Model .....</b>	<b>8</b>
CSEvent Object Specific Methods (Client-Side) .....	9
CSClient Object Specific Methods (Client-Side) .....	9
CSForm Object Specific Methods (Client-Side) .....	12
CSField Object Specific Methods (Client-Side) .....	18
CSChoices Object Specific Methods (Client-Side) .....	28
<b>LiquidOffice Server-Side JavaScript Event Model.....</b>	<b>32</b>
Form Level Events (Server Side) .....	32
<b>LiquidOffice Server-Side JavaScript Object Model .....</b>	<b>36</b>
CSEvent Object Specific Methods (Server Side) .....	36
CSServer Object Specific Methods (Server Side) .....	38
CSForm Object Specific Methods (Server Side) .....	41
CSField Object Specific Methods (Server Side) .....	52
CSChoices Object Specific Methods (Server Side) .....	58
CSUser Object Specific Methods (Server Side) .....	62
CSRRoute Object Specific Methods (Server Side) .....	68
<b>Utility Objects (Server Side) .....</b>	<b>78</b>
<b>File Access .....</b>	<b>80</b>
CSFile Object Specific Methods (Server Side) .....	80

---

CSFileReader Object Specific Methods (Server Side) .....	83
CSFileWriter Object Specific Methods (Server Side) .....	89
<b>Email Access .....</b>	<b>93</b>
CSMail Object Specific Methods (Server Side) .....	93
<b>Database Access .....</b>	<b>96</b>
CSDB Object Specific Methods (Server Side) .....	96
CSConnection Object Specific Methods (Server Side) .....	99
CSRowSet Object Specific Methods (Server Side) .....	105
CSColumn Object Specific Methods (Server Side) .....	121
<b>Common Script Entry Points (Server Side) .....</b>	<b>128</b>
Form Level Common Events .....	129
<b>Response Pages .....</b>	<b>134</b>
URL Response Page .....	134
Custom URL Response Page .....	135
Example Custom Response Page .....	136
LiquidOffice Form Response Page .....	138
<b>Server Object/Method Availability .....</b>	<b>139</b>
Event Availability Table -- Events called for these actions .....	139

---

# JavaScript Editor Object/Event Models

The Script Editor in LiquidOffice Form Designer is designed for users who have a basic understanding of scripting languages, JavaScript, and programming concepts.

The Script Editor allows you to edit and write script based on events and objects contained in a form. Use the methods described in this section as a reference for writing script for forms using the LiquidOffice JavaScript Object Model (LJOM). This section provides methods and descriptions for both the Client-Side and Server-Side LiquidOffice JavaScript Object Models. It also provides a reference for Server-Side utility objects and methods that allow access to the server's file system, email, and database connections.

For a complete overview of the LiquidOffice Form Designer Script Editor see the LiquidOffice Form Designer online help.

For a complete JavaScript reference, see *JavaScript: The Definitive Guide, Third Edition* by David Flanagan. (Flanagan, David. *JavaScript: The Definitive Guide, Third Edition*. Sebastopol, CA. O'Reilly & Associates, Inc., 1998.)

## LiquidOffice Client-Side JavaScript Event Model

JavaScript written using LiquidOffice is event driven. That is, script will not run until some event triggers it (like a button being clicked). Events are broken down into two categories for client script:

- Form Level Events
- Field Level Events

### Form Level Events

---

#### **CSForm\_OnLoad()**

Description:

Called when the form is loaded. CSEvent.getTarget() returns CSForm object.

Example:

```
function CSForm_OnLoad()
{
    CSClient.alert( "The form is now open." );
}
```

---

### **Boolean CSForm\_OnSubmit()**

Description:

Called when a form is about to be submitted. User must return true if it is ok to submit the form - false otherwise. If the user returns false, the form will not be submitted. When submission is terminated, no message box will be displayed automatically, this is the responsibility of the script author.  
CSEvent.getTarget() returns CSForm object.

Returns:

**Boolean**

Returns `true` on success or `false` on failure.

Example:

```
function CSForm_OnSubmit(){
    if (CSClient.alert("Submit?", 1, 2) == 1){
        return true;
    } else {
        CSClient.alert("Your form will not submit.");
        return false;
    }
}
```

## **Field Level Events**

---

### **<Field\_Name>\_OnFocus()**

Description:

Called when the field receives the focus. Use CSEvent to access event information. CSEvent.getTarget() will returns the CSField object associated with this event. Do not set focus to another field from within this event.

Example:

```
function FName_OnFocus()
{
    CSClient.alert("First Name has the focus.");
}
```

---

### **<Field\_Name>\_OnBlur()**

Description:

Called when the field loses the focus. Use CSEvent to access event information. CSEvent.getTarget() returns the CSField object associated with this event.

Example:

```
function FName_OnBlur()
{
    CSClient.alert( "First Name has lost the focus." );
}
```

---

### **<Field\_Name>\_OnButtonClick()**

Description:

Called when this field is clicked. Use CSEvent to access event information. CSEvent.getTarget() returns the CSField object associated with this event.

Example:

```
function Button1_OnButtonClick()
{
    CSClient.alert("You just clicked Button1.");
}
```

---

### **<Field\_Name>\_OnChange()**

Description:

Called when the field value is changed by the user. Use CSEvent to access event information. CSEvent.getTarget() returns the CSField object associated with this event.

Example:

```
function FName_OnChange()
{
    CSClient.alert( "The First Name field has changed." );
}
```

---

## LiquidOffice Client-Side JavaScript Object Model

Once an event occurs, any script associated with that event will run. The LiquidOffice JavaScript Object Model (LJOM) maps the LJOM methods written in the LiquidOffice JavaScript Editor to the client specific object model defined by each client. For example, to obtain a field's value using the LJOM the user would use the following command:

```
CSForm.getField( <Field_Name> ).getValue();
```

For a form published to PDF, the LJOM would map this command to Adobe's Acrobat Forms JavaScript Object Specification (AFJOS) as such:

```
this.getField( <Field_Name> ).value;
```

For a form published to DHTML, the command would be mapped in this way:

```
document.form.element[ <Field_Name> ];
```

The LJOM contains methods rather than properties. This is because methods are the only way to map LJOM to the client specific object model.

Server Script is a direct integration of the scripting engine rather than just a wrapper.

The purpose of script is to inspect or modify some object on the form. The global objects available in client script are:

- CSEvent
- CSClient
- CSForm

Additionally, the following objects can be accessed through CSForm:

- CSField
- CSChoices

The CSEvent object is used to access information related to the most recent event that has occurred. The CSClient object is used to access features specific to the client application. The CSForm object is the form itself. There is only one. CSField objects are any data collection field or button contained in CSForm. CSChoices objects are the individual selections that make up a CSField object ( list box, drop list, and combo-box only).

---

## CSEvent Object Specific Methods (Client-Side)

The methods described below are used for accessing information related to the most current event that has taken place.

### ***Object getTarget()***

**CSField** or **CSForm** -- Depending on the entry point.

Description:

Used to access the object that triggered an event. This will return a CSForm object for form entrypoints: CSForm\_Load() and CSForm\_OnSubmit(). This will return CSField for field entrypoints:

<Field\_Name>\_OnFocus(), <Field\_Name>\_OnBlur(), <Field\_Name>\_OnButtonClick(), and  
<Field\_Name>\_OnChange().

Returns:

**CSField** or **CSForm** -- Depending on the entry point.

Returns the field object that triggered the event or the form.

Example:

```
CSClient.alert( "Event triggered by the " + CSEvent.getTarget().getName() + " field." )
```

---

## CSClient Object Specific Methods (Client-Side)

The methods described below are used to access client application specific functionality and information.

### ***Number alert( String sMsg, Number iBtns, Number iIcon )***

Description:

Displays a message box. sMsg is the message text displayed. iBtns specifies which buttons will be available to the user on the message box. iIcon specifies what graphical icon will be displayed on the message box. iIcon is ignored by HTML forms.

Arguments:

#### **String - sMsg**

String to display in message box.

#### **Number - iBtns**

Several different groups of buttons can be on the message box depending on the client.

Adobe Acrobat	
Buttons	iBtns Value
OK	0
OK, Cancel	1
Yes, No	2
Yes, No, Cancel	3

---

DHTML	
Buttons	iBtns Value
OK	0
OK, Cancel	1

#### Number - iIcon

Icons can be displayed in the message box as well but only in PDF forms. In forms exported to HTML, this is ignored.

Adobe Acrobat	
Icon	iIcon Value
Error (default )	0
Warning	1
Question	2
Status	3

DHTML	
Icon	iIcon Value
<u>NOT SUPPORTED</u> <u>ARGUMENT IGNORED</u>	

---

Returns:

**Number**

Returns the number of the button pressed.

Button Pressed	Return Value
OK	1
Cancel	2
No	3
Yes	4

DHTML	
Button Pressed	Return Value
OK	1
Cancel	0

Example:

```
function CSForm_OnSubmit()
{
if ( CSClient.alert( "Submit the form data?", 1, 2 ) == 1 )
{
    return true;
}
else
{
    CSClient.alert( "Form Submission Has Been Canceled!", 0, 3 );
    return false;
}
}
```

***String prompt( String sMsg, String sDefVal )***

Description:

Displays a message box with an edit box the user can enter some text into. The edit box can have some text in it by default (sDefVal). This message box has OK and Cancel buttons.

Arguments:

**String - sMsg**

Message to display. Should give the user some indication of what input is expected from them.

---

**String - sDelVal**

Default value to prefill in the edit box.

Returns:

**String**

Returns the text entered in the edit box or `null` if the Cancel button was pressed.

Example:

```
sTxt = CSClient.prompt( "Please enter your first name.", "Jane Doe" );
```

```
CSForm.getField( "FName" ).setValue( sTxt );
```

---

**String getAgent()**

Description:

Identifies the client user agent type (Acrobat, Netscape, Internet Explorer) and version that the form is being viewed in.

Returns:

**String**

Returns a string specifying the user agent type and version. Exact value will depend on agent.

Example:

```
CSClient.alert( "The user client is " + CSClient.getAgent() );
```

## CSForm Object Specific Methods (Client-Side)

The methods described below are used at the form level for accessing form specific data including the form's title, unique ID, and location. Additionally, each individual field object is referred to via the CSForm object.

---

**Boolean getFinalized()**

Description:

Gets the state of the form. If it is a finalized form no longer active, then this method returns true. This method is available in the server entry point `CSForm_OnOpen` and the client entry point `CSForm_OnLoad`. This function will return true when an archived form is opened.

Returns:

**Boolean**

True if the form is finalized (went through routing completed or is a non-routing form that was submitted). False otherwise.

Example:

```
if( CSForm.getFinalized() )
    CSClient.alert( "This form has been 'Finalized'." + "\r\n" +
```

---

```
"All fields are now Read Only." );
```

---

### **String getLanguageCode()**

Description:

Gets the language code of the language for which the form was designed. The form language is set on the Form Properties page in LiquidOffice Designer for each form. Language codes are those specified in ISO 639.

Returns:

**String**

Returns a two-character language code as String.

Example:

```
CSClient.alert( "Form Native Language Code: " + CSForm.getLanguageCode() );
```

---

### **String getLJOMVersion()**

Description:

Specifies the version of LiquidOffice JavaScript Object Model that was published with the form. The entire LJOM is embedded within the published form.

Returns:

**String**

Returns a String identifying the version of LiquidOffice JavaScript Object Model that was published with the form.

Example:

```
CSClient.alert( "Object Model Version " + CSForm.getLFJOMVersion() );
```

---

### **String getPubType()**

Description:

Returns the format that the current form is in ( "PDF", "HMTL", etc. ).

Returns:

**String**

Returns a String identifying the format the current form was published to.

Example:

```
CSClient.alert( "This form was published to " + CSForm.getPubType() + " format." );
```

---

### **String getTitle()**

Description:

Gets the title of the current form.

Returns:

**String**

Returns the title of the form as specified in LiquidOffice Designer's Form Properties dialog box.

Example:

```
CSClient.alert( "Form Title: " + CSForm.getTitle() );
```

---

### **String getID()**

Description:

Gets the form GUID of the current form. This is a unique id for the form generated by the server that does not change with each revision of the form. This will be a blank string during publish preview. This will not return meaningful data until after the form is published.

Returns:

**String**

Returns the form's unique identification number.

Example:

```
CSClient.alert( "Form ID: " + CSForm.getID() );
```

---

### **String getLocation()**

Description:

Get the URL or file path of the form. If the form has been published then its URL is returned, otherwise the full UNC path of the file will be returned. Both HTML and PDF return the current location during preview, though in different formats.

Returns:

**String**

Returns the URL or file path of the form. If the form has been published then its URL will be returned, otherwise the full UNC path of the file will be returned.

Example:

```
CSClient.alert( "Form Location: " + CSForm.getLocation() );
```

---

### **Number getNumberOfFields()**

Description:

Returns the number of fields on this form.

Returns:

#### **Number**

Returns the number of fields on the form including hidden and system fields.

Example:

```
CSClient.alert( "Number Of Fields: " + CSForm.getNumberOfFields() );
```

---

### **submit( *String sURL* )**

Description:

Submits the form data to sURL.

Arguments:

#### **String sURL**

Location to POST form data to. This location might be a CGI script. This method is not for use with forms published to a LiquidOffice Server.

Example:

```
CSForm.submit( "http://www.Cardiff.com/LiquidOffice/test.asp" );
```

---

### **reset()**

Description:

Resets the fields on the form to their default values.

Example:

```
CSForm.reset();
```

---

### **setFocus( *String Field\_Name* )**

Description:

Sets focus to the specified field.

Arguments:

#### **String Field\_Name**

Name of the field to receive focus.

Example:

```
CSForm.setFocus( "Entry1" );
```

---

**CSField getField( String Field\_Name )**

Description:

Returns the field object with the given name.

Arguments:

**String Field\_Name**

Name of the field.

Returns:

**CSField**

Returns a CSField object or null if not found.

Example:

```
CSClient.alert( "Value of Entry1 Field is: " + CSForm.getField("Entry1").getValue() );
```

---

**CSField getField( Number Field\_Index )**

Description:

Returns the field object with the given index value.

Arguments:

**Number Field\_Index**

Index of the field. Field indices are zero based. The first field has an index of 0.

Returns:

**CSField**

Returns a CSField object or null if not found.

Example:

```
for ( var i = 0; i < CSForm.getNumberOfFields(); i++ ){
    MsgTxt += CSForm.getField( i ).getName() + "\r";
}
CSClient.alert( MsgTxt );
```

---

### ***Boolean setResponseURL( String sURL )***

Description:

Used to direct the form filler to an alternate response page after submitting the form. This can be any URL the user has access to through their web browser including other LiquidOffice Forms. Setting the response page to a blank string will cause the default response page to be used. Can be called in OnSubmit.

Arguments:

#### ***String sURL***

URL where the user will be taken after submitting the form.

Returns:

#### ***Boolean***

Returns true on success or false on failure.

Example:

```
function CSForm_OnSubmit()
{
    CSForm.setResponseURL("http://www.Cardiff.com");
}
```

---

### ***String getResponseURL()***

Description:

Gets the currently set response URL. If no response URL has been set, a blank string is returned. In this case, the default response page gets used. Can be called in OnSubmit.

Returns:

#### ***String***

Returns the URL set using CSForm.setResponseURL or blank String if not set.

Example:

```
function CSForm_OnSubmit()
{
    if( CSForm.getResponseURL() == "" ){
        CSClient.alert("No Custom Response URL Has Been Set!");
        return false;
    }
    return true;
}
```

---

## CSField Object Specific Methods (Client-Side)

The methods listed below are used at the field level to access individual field properties such as the field's name, value, and type.

---

### ***String getName()***

Description:

Gets the field's name.

Returns:

**String**

Returns field name as **String**.

Example:

```
function Button1_OnClick()
{
    var msg = "Fields:" + "\r\n";
    for( var i = 0; i < CSForm.getNumberOfFields(); i++ )
        msg += i + ":" + CSForm.getField(i).getName() + "\r\n";
    CSClient.alert( msg );
}
```

---

### ***String getDescription()***

Description:

Gets the description of the field as specified in the Field Description property in LiquidOffice Designer. If the description is blank, the Form Designer will prefill this value with the field's label.

Returns:

**String**

Returns the name of the field as specified in the Field Description property when the field was defined using LiquidOffice Designer.

Example:

```
function Button1_OnClick()
{
    var msg = "Fields:" + "\r\n";
    for( var i = 0; i < CSForm.getNumberOfFields(); i++ )
        msg += i + ":" + CSForm.getField(i).getDescription() + "\r\n";
    CSClient.alert( msg );
}
```

---

### **String getValue()**

Description:

Gets the value of the field. If multiple selection list box - returns tab separated list. For checkbox, returns 1 if checked, 0 if not checked.

Returns:

**String**

Returns the value of the field as String.

Example:

```
function paymentType_OnChange()
{
    var f = CSForm.getField("paymentType");

    if( f.getValue() == "Other" )
        CSForm.getField("Other Payments").setRequired( true );
}
```

---

### **setValue( String New\_Value )**

Description:

Sets the value of the field programmatically. If you call setValue() for a multi-select list box, the matching value will be selected and the rest cleared.

Arguments:

**String New\_Value**

New value assigned to the field.

Example:

```
function clearShippingButton_OnButtonClick()
{
    CSForm.getField("shipToName").setValue("");
    CSForm.getField("shipToStreet").setValue("");
    CSForm.getField("shipToCity").setValue("");
    CSForm.getField("shipToState").setValue("");
    CSForm.getField("shipToZip").setValue("");
}
```

---

## **Array getValues()**

Description:

Gets the currently selected values of the field as an array of strings. It will always return at least one value (possibly a blank string), except in the case of a multi-select list box, in this case it can return a zero length array if no values are selected.

Returns:

### **Array**

Returns an array of values for the specified field. If only one value is defined ( as will be for most field types ) then the array will have only one element with that value. In the case of a multiple select list box with nothing selected, a zero length array will be returned.

Example:

```
function List10_OnChange()
{
    CSClient.alert( "You selected the following options: " + "\r\n" +
        CSForm.getField("List10").getValues() );
}
```

Notes:

Acrobat 4.05 does not support multiple-select list boxes.

---

## **setValues( *Array newValues* )**

Description:

Sets the values of a multi-select list box. Each item in the listbox that matches a value in the array gets selected. Each value in the array that does not match an item in the listbox is ignored. If the field is not a multi-select list box, then the first value in the array is used as the value for the field.

Arguments:

### **Array - newValues**

Each item in the list box that matches a value in the array gets selected. Each item in the list box that does not match a value in the array is ignored.

Example:

```
function Button1_OnButtonClick()
{
    var vals = new Array("one","three","five");

    CSForm.getField("List10").setValues( vals );
}
```

Notes:

Adobe Acrobat 4.0 and Reader 4.0 do not support multiple choice list boxes.

---

### **CSChoices getChoices()**

Description:

Gets the CSChoices object which contains all of the choices in the list box that a user may select.

Returns:

#### **CSChoices**

Returns a CSChoices object. Returns null if the field is not "List", "Combo", or "Drop".

Example:

```
function Button1_OnButtonClick()
{
    var items = CSForm.getField("List10").getChoices();

    CSCClient.alert( "List10 has " +
                    items.getCount() +
                    " items to choose from." );
}
```

---

### **String getType()**

Description:

Gets the type of field.

Returns:

#### **String**

Returns a String value specifying the field type. Possible return values are: "Button", "Text", "Signature", "Check", "Radio", "List", "Combo", or "Drop".

Example:

```
CSCClient.alert( "Entry1 is of type " + CSForm.getField("Entry1").getType() );
```

---

### **Boolean isReadOnly()**

Description:

Gets the read only property for this field. Indicates whether this field is editable by the user.

Returns:

#### **Boolean**

Returns true if the field is read only or false if not.

Example:

```
CSCClient.alert( "Entry1 Read Only Setting: " +
                  CSForm.getField("Entry1").isReadOnly() );
```

---

### **setReadOnly( Boolean bReadOnly )**

Description:

Sets the read only attribute of a field.

Arguments:

#### **Boolean bReadOnly**

Pass true to set the field to read only or false to allow editing.

Example:

```
CSForm.getField( "Supervisor" ).setReadOnly( true );
```

---

### **BooleanisRequired()**

Description:

Specifies whether a field is required for submission or not.

Returns:

#### **Boolean**

Returns true if the field is required or false if not.

Example:

```
function paymentType_OnChange()
{
    var f = CSForm.getField("paymentType");

    if( f.getValue() != "Other" )
        if( CSForm.getField("Other Payments").isRequired() )
            CSForm.getField("Other Payments").setRequired( false );
}
```

---

### **setRequired( Boolean bRequired )**

Description:

Sets the Required property of a field. Determines whether the user is required to fill this field before submitting the form. Does not apply to fields of type "Check".

Arguments:

#### **Boolean - bRequired**

Pass true to make field required or false to make it optional.

Example:

```
function paymentType_OnChange()
{
    var f = CSForm.getField("paymentType");

    if( f.getValue() == "Other" )
        CSForm.getField("Other Payments").setRequired( true );
}
```

---

### **Boolean isHidden()**

Description:

Gets the visible property. Indicates whether a field is visible or not.

Returns:

#### **Boolean**

Returns true if the field is hidden or false if it is visible.

Example:

```
function toggleHidden_OnButtonClick()
{
    var f = CSForm.getField("Entry1");

    if( f.isHidden() )
        f setHidden( false );
    else
        f setHidden( true );
}
```

---

**setHidden( Boolean bHidden )**

Description:

Sets the visible attribute of a field.

Arguments:

**Boolean - bHidden**

Pass true to set the field to hidden or false to make the field visible.

Example:

```
function toggleHidden_OnButtonClick()
{
    var f = CSForm.getField("Entry1");

    if( f.isHidden() )
        f.setHidden( false );
    else
        f.setHidden( true );
}
```

---

**String getTextColor()**

Description:

Specifies the color of the text within a field.

Returns:

**String**

Returns the color of the text as a String in the form of "RRGGBB", where RR, GG, and BB are hexadecimal values representing red, green, and blue color values. (e.g. the value "0000FF" represents blue)

Example:

```
CSClient.alert( "Entry1 Text Color: " +
    CSForm.getField("Entry1").getTextColor() );
```

---

### **setTextColor( String sColor )**

Description:

Sets the text color attribute of a field.

Arguments:

#### **String - sColor**

Pass a String in the form of "RRGGBB" where RR, GG, and BB are hexadecimal values representing red, green, and blue color values. (e.g. the value "0000FF" represents blue)

Example:

```
CSForm.getField("Entry1").setTextColor("FF0000");
```

---

### **String getFillColor()**

Description:

Specifies the background color of a field.

Returns:

#### **String**

Returns the color of the background as a String in the form of "RRGGBB", where RR, GG, and BB are hexadecimal values representing red, green, and blue color values. If a field's background is transparent then "transparent" will be returned. (e.g. the value "0000FF" represents blue)

Example:

```
CSClient.alert( "Entry1 Fill Color: " +
    CSForm.getField("Entry1").getFillColor() );
```

---

### **setFillColor( String sColor )**

Description:

Sets the background color attribute of a field.

Arguments:

#### **String - sColor**

Pass a String in the form of "RRGGBB" where RR, GG, and BB are hexadecimal values representing red, green, and blue color values. To set a field's background to transparent, use "transparent". (e.g. the value "0000FF" represents blue)

Example:

```
CSForm.getField("Entry1").setFillColor("0000FF");
```

---

***Boolean* **isPassword()****

Description:

Gets a field's password attribute. When a field's password property is on, all characters entered will be masked with asterisks. Upon submission, the actual data entered will be sent to the server. This can only be used on fields of type "Text".

Returns:

**Boolean**

Returns true if the field's value will be masked or false if not.

Example:

```
function pass_OnChange()
{
    if( CSForm.getField("pass").isPassword() )
        CSClient.alert("This field's value has been masked for your protection.");
}
```

---

***Boolean* **isMultiline()****

Description:

Indicates whether a text field can accept carriage returns or not. This can only be used on fields of type "Text".

Returns:

**Boolean**

Returns true if the field is multiline or false if not.

Example:

```
function details_OnFocus()
{
    if( CSForm.getField("details").isMultiline() )
        CSClient.alert("You can enter multiple lines into this field.");
}
```

---

### **Number getMaxLength()**

Description:

Gets the maximum length of fields of type "Text" as specified in the Field's Maximum Length property in LiquidOffice Designer.

Returns:

#### **Number**

Returns the maximum length a field will accept or -1 on failure.

Example:

```
CSClient.alert( "Entry1 Max Length: " +  
    CSForm.getField("Entry1").getMaxLength() );
```

---

### **setFocus()**

Description:

Moves the focus to the field.

Example:

```
function Entry2_OnBlur()  
{  
    var e1 = CSForm.getField("Entry1");  
    var e2 = CSForm.getField("Entry2");  
  
    if( e2.getValue() == e1.getValue() ){  
        CSClient.alert("Please enter a different value.");  
        e2.setFocus();  
    }  
}
```

---

## CSChoices Object Specific Methods (Client-Side)

The methods listed below are used at the choice object level to access individual choice object properties such as the display value, export value, and selection status.

### ***Number getCount()***

Description:

Gets the number of items in a "List", "Combo", or "Drop" field.

Returns:

#### **Number**

Returns the number of items in the list.

Example:

```
function Button1_OnButtonClick()
{
    var items = CSForm.getField("List10").getChoices();

    CSClient.alert( "List10 has " +
                    items.getCount() +
                    " items to choose from." );
}
```

### ***String getAt( Number n, Boolean bDisplayValue )***

Description:

Gets the display or export/storage value for the n-th item in a "List", "Combo", or "Drop" field.

Arguments:

#### **Number - n**

Index of the item. Item indices are zero based. The first item in the list has an index of 0. n must be between 0 and getCount() - 1.

#### **Boolean - bDisplayValue**

Pass true to return the item's display value or false to return the export/storage value.

Example:

```
function ListItemsButton_OnButtonClick()
{
    var items = CSForm.getField("List10").getChoices();
    var msg = "List Items:" + "\r\n";

    for( var i = 0; i < items.getCount(); i++ )
        msg += items.getAt(i, true) + "\r\n";
}
```

---

```
msg += items.getAt(i,true) + "\r\n";
```

```
    CSClient.alert( msg );
}
```

Notes:

Adobe Acrobat 4.05 does not support display values. Only the storage value is available.

---

### **insertAt(*Number n*, *String exportValue*, *String displayValue*)**

Description:

Inserts a new item in the n-th position of fields of type "List", "Combo", and "Drop".

Arguments:

**Number - n**

Index of the new item. Passing 0 will insert the new item at the top of the list, -1 - at the bottom of the list.

**String - exportValue**

Export Value for the new list item.

**String - displayValue**

Display Value for the new list item. Passing null causes exportValue to be used.

Example:

```
function AddItem_OnButtonClick()
{
    var items = CSForm.getField("List10").getChoices();
    var nDispVal = CSClient.prompt("Enter Display Value: ", "new val");
    var nStorVal = CSClient.prompt("Enter Storage Value: ", "newVal");

    items.insertAt( -1, nStorVal, nDispVal );
}
```

Notes:

Adobe Acrobat 4.05 does not support display values. Only the storage value will be used.

---

## **deleteAt( Number n )**

Description:

Removes the n-th item from a "List", "Combo", or "Drop" field.

Arguments:

### **Number - n**

Index of the item to remove. Item indices are zero based. The first item in the list has an index of 0.

Example:

```
function DeleteItem_OnButtonClick()
{
    var items = CSForm.getField("List10").getChoices();
    var idx = CSClient.prompt("Enter number of item to delete: ", "1");

    items.deleteAt( idx );
}
```

---

## **Array getSelections()**

Description:

Gets an array of indices of the selected items in the CSChoices object. All indices are zero based.

Returns:

### **Array**

Returns an Array of indices of the selected items in the CSChoices object. Returns a zero length Array if no items are in the list or null if there was an error.

Example:

```
function List1_OnChange()
{
    var list1 = CSForm.getField("List1").getChoices();
    var list2 = CSForm.getField("List2").getChoices();

    list2.setSelections( list1.getSelections() );
}
```

---

## **setSelections( *Array* selections )**

Description:

Selects items in the list using an array of item indices.

Arguments:

### **Array - aSelections**

An Array of item indices to select in the list.

Example:

```
function List1_OnChange()
{
    var list1 = CSForm.getField("List1").getChoices();
    var list2 = CSForm.getField("List2").getChoices();

    list2.setSelections( list1.getSelections() );
}
```

---

## LiquidOffice Server-Side JavaScript Event Model

Up to this point all of the methods described have been those that run on the client-side. That is, the scripts written are contained inside the PDF, or HTML file that LiquidOffice Designer created and are then compiled, run, and managed by the client application (Adobe Acrobat, or Web Browser) on the client machine.

The LiquidOffice Designer JavaScript Editor also allows script to be entered into several server-side entry points. These Server scripts will be published with the form and will be compiled and run [on the server](#). Many of the objects and methods available for writing client-side script are also available for writing server-side script but there are some limitations and differences.

The section below describes the objects and methods available for writing server-side scripts using Form Level Events.

See “[Server Object/Method Availability](#)” on page 139 for a table that shows when each method can be used.

### Form Level Events (Server Side)

Just like the client-side scripts, server-side scripts are not run until some event triggers it.

---

#### **CSForm\_OnOpen()**

Description:

Called just before the form is sent to the user. Typically used to modify form data before the user views the form being served up. CSServer, CSForm, and CSEvent objects are available in this entry point.

Returns:

**Boolean**

Return true to continue opening or false to cancel.

Example:

```
function CSForm_OnOpen()
{
    CSServer.log( "Form Open: " + CSForm.getID() );
```

---

### **CSForm\_OnSubmit()**

Description:

Called when the user clicks on the Go button in a form with one of these actions selected: Submit, Approve, Transfer or Reject. Typically used as a final check on the integrity of the form. This entry point is called whether the form is set up for routing or not. CSServer, CSForm, and CSEvent objects are available in this entry point. Use CSForm.setStatusCode() to fail the submission. When setting this to a negative number, typically you will set the status message (CSForm.setStatusMsg()) to give the user a better idea of what went wrong with the submission

Example:

```
function CSForm_OnSubmit()
{
    CSServer.log( "Form Submit: " + CSForm.getID() );
}
```

---

### **CSForm\_OnRoutePage()**

Description:

Called just before the Routing Page is presented to the user. CSServer, CSRoute, and CSEvent objects are available in this entry point. CSForm is available if the action that triggered this event (CSRoute.getStatus()) is "Submit", "Transfer", or "Approve". CSForm is not available if the action is a "Transfer", "Reject" or "Redirect". If you call CSRoute.setRoutingComplete( true ) or set a valid user in the RouteTo destination with CSRoute.setRouteTo() and call setShowPage( false ), the routing page will be bypassed and will not be presented to the user. This can be used to automate certain portions of a route

Example:

```
function CSForm_OnRoutePage()
{
    CSServer.log( "Route Page Event For Form: " + CSForm.getID() );
}
```

---

### **CSForm\_OnRoute()**

Description:

Called just before the form is routed and after the user has made selections on the routing page if one was presented. CSServer, CSRoute, and CSEvent objects are available in this entry point. CSForm is available if the action that triggered this event (CSRoute.getStatus()) is "Submit" or "Approve". CSForm is not available if the action is "Transfer", "Reject", or "Redirect". In this entry point, the routing destination or completion status can be changed from what the user set on the routing page.

Example:

```
function CSForm_OnRoute()
{
    CSServer.log( "Form Route Event For Form: " + CSForm.getID() );
}
```

---

### **CSForm\_OnExport()**

Description:

Called just before data for the form is stored in the internal database and exported to the Connect Agents defined for the form. The data for the form can be transformed for storage in this entry point. CSServer, CSForm, and CSEvent objects are available in this entry point.

Example:

```
function CSForm_OnExport()
{
    CSServer.log( "Form Export: " + CSForm.getID() );
}
```

---

### **CSForm\_OnValidateLookup()**

Description:

The OnValidateLookup event will occur every time a Database Lookup or Field Validation takes place. Database Lookups can run at Form Load time or as a result of a user Tabbing out of a field or clicking a button. Field Validations can run either when tabbing out of the field being validated or when the 'Go' button is clicked to submit the form. OnValidateLookup will always occur before the OnSubmit entry point is called. CSServer, CSForm, and CSEvent objects are available in this entry point. In this entry point, form data can be changed effectively altering the behavior of a pre-defined database lookup. Additionally data can be validated. Use CSForm.setStatusCode() to reject the validation. When setting a negative status code, typically you will set the status message (CSForm.setStatusMsg()) so the user knows what is wrong with the form and set the focus (CSForm.setFocus()) to the field that needs correction.

---

Example:

```
function CSForm_OnValidateLookup()
{
    CSServer.log( "Form Validate/Lookup: " + CSForm.getID() );
}
```

See also “[Common Script Entry Points \(Server Side\)](#)” on page 128.

---

### **CSForm\_OnComplete()**

Description:

This is the last entry point to run during an event life cycle. OnComplete runs after all routing and exports have been completed. This is the only entry point from which CSForm.mergePDF() can be called where it will include all routing notes and trace data.

Example:

```
function CSForm_OnComplete()
{
    var msg = "**** PDF Form Merge Upon Form Completion ****" + "\r\n" + "Merge File Path: ";
    msg += CSForm.mergePDF( true, true );

    CSServer.log( msg );
}
```

---

## LiquidOffice Server-Side JavaScript Object Model

Just like the client-side object model, the server-side object model consists of objects that allow the script to inspect or modify form objects. In addition to having some of the same objects and methods as the client-side object model, the server-side object model also provides some additional objects and methods that allow access to the server's file system, email, and database connections.

It is important to note that while some of the server-side objects and methods share the same names and functionality as their client-side counterparts, there are certain limitations and differences that must be considered.

The server-side object model consists of these global objects:

- CSEvent
- CSServer
- CSForm
- CSRoute

Additionally, the following objects can be accessed through the global objects:

- CSField
- CSChoices
- CSUser

### CSEvent Object Specific Methods (Server Side)

The methods described below are used for accessing information related to the most current event that has taken place.

---

#### *Object* **getTarget()**

Description:

Used to access the object that triggered the event. This will return a CSField object for field level events such as CSForm\_OnValidateLookup() and a CSForm object for form level events.

Returns:

**CSField** or **CSForm** -- Depending on the entry point.

Returns the field object that triggered the event or the form object.

---

Example:

```
function CSForm_OnValidateLookup()
{
    var target = CSEvent.getTarget();
    if( target.getName() == "LookupButton1" )
        if( CSForm.setStatusMsg("Please confirm your information is correct,")
            target.setHidden( true );
}
```

---

### ***string getAction()***

Description:

Gets the action that triggered the event. This method is only valid in the CSForm\_OnValidateLookup() entry point. For this function, when the user Approves or Transfers a form, it is considered a Submit if done from within the form.

Returns:

**String**

Returns the name of the action that triggered the event: one of "Buttonclick", "Tabout" or "Submit".

Example:

```
function CSForm.OnValidateLookup()
{
    if( CSEvent.getAction() == "Tabout" )
        if( CSEvent.getTarget().isPassword() )
            target.setReadOnly( true );
}
```

---

## CSServer Object Specific Methods (Server Side)

The methods listed below are used at the server level to access server objects and properties such as the log file, version number, and user objects.

---

### **log( String message )**

Description:

Write a message to the log file.

Arguments:

#### **String message**

Text to add in the server log. This message will be preceded automatically by a system generated time stamp.

Example:

```
CSServer.log("**** Writing To Server Log File ****");
```

---

### **String getVersion()**

Description:

Gets the version number of the server.

Returns:

#### **String**

Returns the version LiquidOffice Server version as a String.

Example:

```
CSServer.log("LiquidOffice Server Version: " + CSServer.getVersion() );
```

---

### **CSUser getCurrentUser()**

Description:

Gets the current (logged in) user object.

Returns:

#### **CSUser**

Returns a CSUser object for the currently logged in user or null if the user is unknown (i.e. anonymous).

Example:

```
function CSForm_OnOpen()
{
    var user = CSServer.getCurrentUser();

    CSForm.getField("FName").setValue( user.getFirstName() );
    CSForm.getField("LName").setValue( user.getLastName() );
```

---

```
        CSForm.getField("email").setValue( user.getEmailAddress() );
    }
```

---

### ***CSUser getUser( String UserID )***

Description:

Gets the user object for the specified user ID.

Arguments:

#### **String UserID**

Pass in the user id of the user to look up.

Returns:

#### **CSUser**

Returns a CSUser object for the specified user ID or null if not found.

Example:

```
CSForm_OnValidateLookup()
{
    var uID = CSForm.getField("lookupUserID").getValue();
    var uFN = CSForm.getField("UserFirstName");
    var uLN = CSForm.getField("UserLastName");

    uFN.setValue( CSServer.getUser( uID ).getFirstName() );
    uLN.setValue( CSServer.getUser( uID ).getLastName() );
}
```

---

### ***Array getUsersByName( String first, String last )***

Description:

Gets an array of CSUser objects for users with matching first and/or last name. Uses exact String match search. First name or last name may be omitted to search on only one or the other.

Arguments:

#### **String - first**

First name to search for. If null then only last name will be searched for.

#### **String - last**

Last name to search for. If null then only first name will be searched for.

Returns:

#### **Array**

---

Returns an array of CSUser objects for the specified user name. Uses exact match. If the first name is null will lookup only by the last name and vice-versa. Both parameters must be passed. To omit either just pass an empty String ( "" ). Returns 0 length array if there is no match.

Example:

```
CSForm_OnOpen()
{
    var user = CSServer.getCurrentUser();
    var uLN = user.getLastName();

    var aUsrs = CSServer.getUsersByName( null, uLN );

    if( aUsrs.length != 0 ){
        var usrList = CSForm.getField("CommonNameList").getChoices();

        for( var i = 0; i < aUsrs.length; i++ ){
            var uID = aUsrs[i].getUserID();
            var uFirst = aUsrs[i].getFirstName();
            var uLast = aUsrs[i].getLastName();

            usrList.insertAt( -1, uID, uFirst + " " + uLast );
        }
    }
}
```

---

## CSForm Object Specific Methods (Server Side)

The methods described below are used at the form level for accessing form specific data including the form's title, unique ID, and location. Additionally, each individual field object is referred to via the CSForm object.

There are many of the same methods listed here as are under client-side CSForm Object Specific Methods.

---

### **Boolean getFinalized()**

Description:

Gets the state of the form. If it is a finalized form no longer active, then this method returns true. This method is available in the server entry point CSForm\_OnOpen and the client entry point CSForm\_OnLoad. This function will return true when an archived form is opened.

Returns:

**Boolean**

True if the form is finalized (went through routing completed or is a non-routing form that was submitted). False otherwise.

Example:

```
function CSForm_OnOpen()
{
    if( CSForm.getFinalized() )
        CSForm.setStatusMsg( "This form has been 'Finalized'." + "\r\n" +
                            "All fields are now Read Only." );
}
```

---

### **setFocus( String fieldName )**

Description:

Sets the focus to this field. Can be called from OnOpen and OnValidateLookup entry points.

Arguments:

**String - fieldName**

Name of the field to receive focus.

Example:

```
function CSForm_OnOpen()
{
    CSForm.setFocus("FirstName");
}
```

---

### **String getPubType()**

Description:

Gets the format that the current form is in. Currently "PDF" or "HTML".

Returns:

**String**

Returns a String specifying the format that the current form is in. Either "PDF" or "DHTML".

Example:

```
function CSForm_OnSubmit()
{
    if( CSForm.getPubType() == "PDF" )
        var format = "pdf";
    else if( CSForm.getPubType() == "HTML" )
        var format = "crp";

    CSForm.setResponseURL("http://LOServer/lfserver/HRForms/SecondaryForm?DFS__FormType=" +
        format );
}
```

---

### **String getTitle()**

Description:

Gets the title of the form.

Returns:

**String**

Returns a String giving the title of the form.

Example:

```
function CSForm_OnExport()
{
    CSServer.log("Form Export..." + "\r\n"
        "Form: " + CSForm.getTitle() );
}
```

---

### **String getID()**

Description:

Gets the unique identification number of the current form. This is a unique id for the form generated by the server that does not change with each revision of the form.

Returns:

**String**

Returns the form's GUID (Global Unique Identifier) as a String value.

Example:

```
function CSForm_OnExport()
{
    CSServer.log("Form Export..." + "\r\n"
        "FormID: " + CSForm.getID());
}
```

---

### **String getRevID()**

Description:

Gets the form revision number. This is similar to the GUID but changes each time a new revision of the form is published.

Returns:

**String**

Returns the form's revision number as a String value.

Example:

```
function CSForm_OnExport()
{
    CSServer.log("Form Export..." + "\r\n"
        "Form: " + CSForm.getTitle() + "\r\n" +
        "Revision: " + CSForm.getRevID());
}
```

---

### **String getStatus()**

Description:

Gets the form status.

Returns:

**String**

Possible values are "Success", "Delayed", "Incomplete", "Failed", "Severe", "Fatal", and "Postponed".

Example:

```
CSServer.log("Form Status: " + CSForm.getStatus());
```

---

## **Number getStatusCode()**

Description:

Gets the form's status code.

Returns:

### **Number**

These values correspond to the status codes strings returned by getStatus().

Successful status codes are:

- 1 - "Success"
  - 2 - "Delayed"
  - 3 - "Incomplete"
- 

Failure status codes are:

- 1 - reserved for future use.
- 2 - "Failed"
- 3 - "Severe"
- 4 - "Fatal"
- 5 - "Postponed"

Example:

```
CSServer.log("Form Status Code: " + CSform.getStatusCode() );
```

---

## **setStatusCode( Number status )**

Description:

Sets the status code. Form status code may be set to Success or Failed. Setting the status code to Failed stops the form event from processing., the form will not be routed, submitted, or exported.

Arguments:

### **Number - status**

Argument must be one of the valid status codes.

Successful status code:

- 1 - "Success"
- 

Failure status code:

- 2 - "Failed"

---

Example:

```
function CSForm_OnSubmit()
{
    if( CSForm.getField("chkNevermind").getValue() == "1" ){
        CSForm.setStatusMsg("Form submission cancelled.");
        CSForm.setStatusCode(-2);
    }
}
```

---

### ***String getStatusMsg()***

Description:

Gets the status message. This may contain a more informative message than what is contained in `getStatus()`.

Returns:

**String**

Returns the status message as a String value.

Example:

```
CSServer.log("Status Message: " + CSForm.getStatusMsg() );
```

---

### ***setStatusMsg( String message )***

Description:

Sets the status message. When called in `CSForm_OnOpen()` or `CSForm_OnValidateLookup()`, 'message' is presented to the user within the form as a status message. When called in `CSForm_OnSubmit()`, 'message' is presented to the user in an HTML error page following the form. Behavior when called in other entry points is not defined.

Arguments:

**String - message**

Status message.

Example:

```
function CSForm_OnSubmit()
{
    if( CSForm.getField("RequiredName").getValue() == "" ){
        CSForm.setStatusMsg("The Required Name field must be filled.");
        CSForm.setStatusCode(-2);
    }
}
```

---

**Number getNumberOfFields()**

Description:

Gets the number of fields on the form.

Returns:

**Number**

Returns the number of field on the form.

Example:

```
function CSForm_OnOpen()
{
    var nReqFields = 0;

    for( var i = 0; i < CSForm.getNumberOfFields(); i++ )
        if( CSForm.getField(i).isRequired() )
            nReqFields++;

    CSForm.setStatusMsg("There are " + nReqFields + " required fields on this form." );
}
```

---

**String getLanguageCode()**

Description:

Gets the language code of the language for which the form was designed. The form language is set on the Form Properties page in LiquidOffice Designer for each form. Language codes are those specified in ISO 639.

Returns:

**String**

Returns two-character language code.

Example:

```
function CSForm_OnOpen()
{
    CSServer.log("Form Opened. Language Code = " + CSForm.getLanguageCode());
}
```

---

### **String getLocation()**

Description:

Get the URL from where the form was retrieved.

Returns:

#### **String**

Returns the URL where the form was posted as a String value.

Example:

```
function CSForm_OnOpen()
{
    CSServer.log("*** Form Opened ***" + "\r\n" +
        "Form: " + CSForm.getTitle() + "\r\n" +
        "Location: " + CSForm.getLocation());
}
```

---

### **Array getAttachments()**

Description:

Gets an array of strings specifying file pathnames (full paths) for all form attachments. Does not include URL attachments. Can be called from the OnExport entry point and from Custom Export scripts.

Returns:

#### **Array**

Returns an array of String pathnames or a zero length array if there are no attachments.

Example:

```
function CSForm_OnExport()
{
    var atts = CSForm.getAttachments();
    var msg = "*** Form Attachments List ***" + "\r\n" +
        "Form Name: " + CSForm.getTitle() + "\r\n";
    for( var i = 0; i < atts.length; i++ )
        msg += atts[i] + "\r\n";
    CSServer.log( msg );
}
```

---

## **Array getAttachmentURLs()**

Description:

Gets an array of strings specifying URLs for all form URL attachments. Does not include file attachments. Can be called from the OnExport entry point and from Custom Export scripts.

Returns:

### **Array**

Returns an array of String URLs or a zero length array if there are no URL attachments.

Example:

```
function CSForm_OnExport()
{
    var atts = CSForm.getAttachmentURLs();
    var msg = "*** Form URL Attachments List ***" + "\r\n" +
        "Form Name: " + CSForm.getTitle() + "\r\n";

    for( var i = 0; i < atts.length; i++ )
        msg += atts[i] + "\r\n";

    CSServer.log( msg );
}
```

---

## **String getDocument()**

Description:

Gets a PDF representation of the submitted form with data. If a signed PDF is available, the signed PDF is used. Otherwise, the current data for the form is merged into a blank PDF to produce a completed PDF. Once the data is merged onto a PDF, that copy of the PDF is always returned by this function until another form submission takes place (usually by a user routing the form to another user). Function may be called in any server script form entry point where CSForm is defined.

Returns:

### **String**

The full pathname of the signed or merged PDF.

Example:

```
function CSForm_OnExport()
{
    var docPath = CSForm.getDocument();

    CSServer.log("Signed Form Available" + "\r\n" +
```

---

```
"Form: " + CSForm.getTitle() + "\r\n" +
"Path: " + docPath );
}
```

---

### **String mergePDF( Boolean bNotes, Boolean bTrace )**

Description:

Generates a PDF of the current form by merging the most recently submitted data with the blank form template. The signed PDF is never used. If either parameter is set to true, additional information is merged onto the document. Unlike getDocument, this method generates a new PDF each time the method is called. Function may be called in any server script form entry point where CSForm is defined.

Arguments:

**Boolean - bNotes**

If true, add a page to the document with all of the routing notes that were added to the form up until present time.

**Boolean - bTrace**

If true, add a page to the document with the trace history for the form up until present time.

Returns:

**String**

The full pathname of the merged PDF file.

Example:

```
function CSForm_OnExport()
{
    var docPath = CSForm.mergePDF( true, true );

    CSServer.log("Merged Form Available" + "\r\n" +
        "Form: " + CSForm.getTitle() + "\r\n" +
        "Path: " + docPath );
}
```

---

### **CSField getField( String fieldName )**

Description:

Gets the field object with the given name.

Arguments:

**String - fieldName**

Name of the field.

---

Returns:

**CSField**

Returns a CSField object for the field or null if the field is not found.

Example:

```
function CSForm_OnOpen()
{
    var user = CSServer.getCurrentUser();

    CSForm.getField("FName").setValue( user.getFirstName() );
    CSForm.getField("LName").setValue( user.getLastName() );
    CSForm.getField("email").setValue( user.getEmailAddress() );
}
```

---

***CSField getField( Number index )***

Description:

Gets the field object at the given index.

Arguments:

**Number - index**

Index of the field. Field indices are zero based. The first field will have an index of 0.

Returns:

**CSField**

Returns the field object or null if index is out of range. Valid range for index is 0 to getNumberOfFields() - 1.

Example:

```
function CSForm_OnOpen()
{
    var nReqFields = 0;

    for( var i = 0; i < CSForm.getNumberOfFields(); i++ )
        if( CSForm.getField(i).isRequired() )
            nReqFields++;

    CSForm.setStatusMsg("There are " + nReqFields + " required fields on this form.");
}
```

---

### ***String getResponseURL()***

Description:

Gets the currently set response URL. If no response URL has been set, returns a blank string. In this case the default response page is used. Can be called from OnSubmit and OnRoute on the server.

Returns:

**String**

Returns the current response URL.

Example:

```
function CSForm_OnSubmitBefore()
{
    var usr = CSServer.getCurrentUser();
    if( !usr )
        if( CSForm.getResponseURL() == "" )
            CSForm.setResponseURL("http://www.cardiff.com");
}
```

---

### ***setResponseURL( String sURL )***

Description:

Specifies what URL will be displayed to the user after submitting the form. Setting the response URL to a blank string will cause the default response URL to be used. This value is blank by default. Can be called in OnSubmit and OnRoute on the server.

Arguments:

**String - sURL**

URL to direct the user to after submission.

Example:

```
function CSForm_OnSubmit()
{
    CSForm.setResponseURL("http://www.Cardiff.com");
}
```

---

## CSField Object Specific Methods (Server Side)

The methods listed below are used at the field level to access individual field properties such as the field's name, value, and type. Additionally, each individual choice object is accessed via the CSField object.

---

### **String getDefaultDataType()**

Description:

Gets the data type of the field. Used for exports when creating tables.

Returns:

**String**

Returns the data type as a String. Possible return values are "Numeric", "String", "Date", and "Time".

Example:

```
function CSForm_OnOpen()
{
    var msg = "Fields :: Default Data Type" + "\r\n";
    for( var i = 0; i < CSForm.getNumberOfFields(); i++ ){
        var f = CSForm.getField(i);
        msg += i + ":" + f.getName() + " :: " + f.getDefaultDataType() + "\r\n";
    }

    CSForm.setStatusMsg( msg );
}
```

---

### **setFocus()**

Description:

Sets the focus to this field upon returning to the client. Only works in CSForm\_OnOpen() and CSForm\_OnValidateLookup().

Example:

```
function CSForm_OnOpen()
{
    CSForm.getField("FirstName").setFocus();
}
```

---

### **String getName()**

Description:

Gets the name of the field.

Returns:

**String**

Returns the name of the field as a String value.

Example:

```
function CSForm_OnOpen()
{
    var msg = "Fields :: Default Data Type" + "\r\n";

    for( var i = 0; i < CSForm.getNumberOfFields(); i++ ){
        var f = CSForm.getField(i);
        msg += i + ":" + f.getName() + " :: " + f.getDefaultDataType() + "\r\n";
    }

    CSForm.setStatusMsg( msg );
}
```

### **String getDescription()**

Description:

Gets the description of the field as specified in the Field Description property in LiquidOffice Designer.  
If the description is blank, it will get the label of the field.

Returns:

**String**

Returns the description of the field as String.

Example:

```
function CSForm_OnOpen()
{
    var msg = "Fields :: Description" + "\r\n";

    for( var i = 0; i < CSForm.getNumberOfFields(); i++ ){
        var f = CSForm.getField(i);
        msg += i + ":" + f.getName() + " :: " + f.getDescription() + "\r\n";
    }

    CSForm.setStatusMsg( msg );
}
```

---

### **String getValue()**

Description:

Gets the value of the field. If the field is a multiple selection list box the storage values will be presented as a tab separated list. For checkbox, returns "1" if checked or "0" if not checked.

Returns:

**String**

Returns the value of the field as String.

Example:

```
function CSForm_OnOpen()
{
    var msg = "Fields :: Value" + "\r\n";

    for( var i = 0; i < CSForm.getNumberOfFields(); i++ ){
        var f = CSForm.getField(i);
        msg += i + ":" + f.getName() + " :: " + f.getValue() + "\r\n";
    }
    CSForm.setStatusMsg( msg );
}
```

---

### **setValue( String newValue )**

Description:

Sets the value of the field. If you call setValue for a multi-select list box, one value will be selected and the rest cleared.

Arguments:

**String - newValue**

New value of the field.

Example:

```
CSForm.getField("Entry1").setValue("Hello world");
```

---

### **Array getValues()**

Description:

Gets the currently selected values of the field as an array of strings. It will always return at least one value (possibly a blank string), except in the case of a multi-select list box, in this case it can return a zero length array if no values are selected.

---

Returns:

**Array**

Returns an array of values for the specified field. If only one value is defined ( as will be for most field types ) then the array will have only one element with that value. In the case of a multiple select list box with nothing selected, a zero length array will be returned.

Example:

```
CSForm.getField("List2").getValues();
```

---

**setValues( *Array newValues* )**

Description:

Sets the values of a multi-select list box.

Arguments:

**Array - newValues**

Array of Strings, each item in the list box that matches a value in the array gets selected. Each item in the list box that does not match a value in the array is ignored.

Example:

```
function CSForm_OnOpen()
{
    var vals = new Array("one","three","five");

    CSForm.getField("List10").setValues( vals );
}
```

---

**CSChoices getChoices()**

Description:

Gets the CSChoices object which contains all of the choices in the list box that a user may select. Returns null if the field is not of type "List", "Combo", or "Drop".

Returns:

**CSChoices**

Returns the CSChoices object associated with this field.

Example:

```
function CSForm_OnOpen()
{
    var items = CSForm.getField("List10").getChoices();

    CSForm.setStatusMsg( "List10 has " +
        items.getCount() +
        " items to choose from." );
}
```

---

### ***String*** **getType()**

Description:

Gets the type of field.

Returns:

**String**

Possible Values are "Button", "Text", "Check", "Radio", "List", "Combo", and "Drop".

Example:

```
function CSForm_OnOpen()
{
    var msg = "Fields :: Field Type" + "\r\n";

    for( var i = 0; i < CSForm.getNumberOfFields(); i++ ){
        var f = CSForm.getField(i);
        msg += i + ": " + f.getName() + " :: " + f.getType() + "\r\n";
    }

    CSForm.setStatusMsg( msg );
}
```

---

### ***Boolean*** **isRequired()**

Description:

Indicates whether the user is required to fill this field out before submission.

Returns:

**Boolean**

Returns true if the field is required or false if not.

Example:

```
function CSForm_OnOpen()
{
    var msg = "Required Fields" + "\r\n";

    for( var i = 0; i < CSForm.getNumberOfFields(); i++ )
        if( CSForm.getField(i).isRequired() )
            msg += CSForm.getField(i).getName() + "\r\n";

    CSForm.setStatusMsg( msg );
}
```

---

### ***Number*** **getMaxLength()**

Description:

Gets the maximum length of the field as set in the field's Maximum Length property in LiquidOffice Designer.

Returns:

#### **Number**

Returns the maximum number of characters that a field will allow.

Example:

```
function CSForm_OnOpen()
{
    var msg = "Field :: Max Length" + "\r\n";

    for( var i = 0; i < CSForm.getNumberOfFields(); i++ ){
        var f = CSForm.getField(i);

        if( f.getType() == "Text" )
            msg += f.getName() + " :: " + f.getLength() + "\r\n";
        CSForm.setStatusMsg( msg );
    }
}
```

---

## CSChoices Object Specific Methods (Server Side)

The methods listed below are used at the choice object level to access individual choice object properties such as the display value, export value, and selection status.

### ***Number getCount()***

Description:

Gets the number of items in a list box, drop list, or combo box.

Returns:

#### **Number**

Returns the number of items in the list.

Example:

```
function CSForm_OnOpen()
{
    var items = CSForm.getField("List10").getChoices();

    CSForm.setStatusMsg( "List10 has " +
        items.getCount() +
        " items to choose from." );
}
```

---

### ***String getAt( Number n, Boolean bDisplay )***

Description:

Gets the display or export/storage value for the n-th item in the list box, combo box or drop-down list box. CSChoices will include all fixed/dynamically pre-filled choices only on form open. In all other instances, they will include only the selected by user values or will be empty. (Or the changes made by the script to add/remove entries till this point after the form submission).

Arguments:

#### **Number - n**

Index of the item. Must be between 0 and getCount() - 1.

#### **Boolean - bDisplay**

Pass true to return display value or false to return storage/export value.

Returns:

#### **String**

Returns the storage/export value if bDisplay is false or the display value if bDisplay is true of the n-th item in the list/combo drop-downs.

---

Example:

```
function CSForm_OnOpen()
{
    var items = CSForm.getField("List10").getChoices();
    var msg = "List Items:" + "\r\n";

    for( var i = 0; i < items.getCount(); i++ )
        msg += items.getAt(i,true) + "\r\n";

    CSForm.setStatusMsg( msg );
}
```

---

### **insertAt( Number n, String exportValue, String displayValue )**

Description:

Inserts a new item in the n-th position of fields of type "List", "Combo", and "Drop".

Arguments:

**Number - n**

Index of the new item. Passing 0 inserts the item at the top of the list, -1 inserts at the bottom.

**String - exportValue**

Storage/export value of the new item.

**String - displayValue**

Display value of the new item. Passing an empty String will cause the storage/export value to be used.

Example:

```
function CSForm_OnOpen()
{
    // Variable holding Connect Agent Name to use for database connections
    var dataLink = "DB1";

    // Create a connection to the database
    var connection = CSDB.newConnection( dataLink );

    // Check validity of connection -- if null then there's a problem
    if( !connection ){
        // Print an error to the server log
        CSServer.log("*** Database Error ***" + "\r\n" +
            "Unable to establish database connection to " +
```

---

```
        dataLink + "." + "\r\n" +
        "**** End Error ***" );
    return;
}

// Variable holding SQL statement
var sql = "SELECT * FROM \"Accounting Project Codes\";

// Execute the SQL and return a row set
var resultSet = connection.executeSQL( sql );

// Check validity of rowset -- if null then there's a problem
if( !resultSet ){
    // Print an error to the server log
    CSServer.log("**** Rowset Error ***" + "\r\n" +
        "Unable to open rowset. SQL: " + sql + "\r\n" +
        "**** End Error***" );
    return;
}
// Variable holding CSChoices object for List field
var listChoices = CSForm.getField("List1").getChoices();

// Loop through the row set until there are no more rows
while( resultSet.nextRow() ){
    // Variable to hold Project name from current row
    var projectname = resultSet.getValue( "Project" );

    // Add choice to list
    listChoices.insertAt( -1, projectname, projectname );
}

// Close the row set
resultSet.close();

// Close the database connection
connection.close();
}
```

---

## **deleteAt( Number n )**

Description:

Removes the n-th item from a "List", "Combo", or "Drop" field.

Arguments:

### **Number - n**

Index of the item to remove. Item indices are zero based. The first item in the list has an index of 0. n must be between 0 and getCount() - 1.

Example:

```
function CSForm_OnOpen()
{
    var items = CSForm.getField("List1").getChoices();
    while (items.getCount() > 0) {
        items.deleteAt(0);
    }
}
```

---

## CSUser Object Specific Methods (Server Side)

The methods listed below are used at the user object level to access object properties for users such as their email address, UserID, and name.

---

### **String getLanguageCode()**

Description:

Gets the language code of the language the user is currently using. Language code is determined based on the user preferences set on the server. Language codes are those specified in ISO 639.

Returns:

**String**

Returns a two-character language code as String.

Example:

```
function CSForm_OnOpen()
{
    var usr = CSServer.getCurrentUser();

    switch(usr.getLanguageCode()){
        case "en":
            CSForm.setStatusMsg("Good Morning");
            break;
        case "de":
            CSForm.setStatusMsg("Guten Tag");
            break;
    }
}
```

---

### **String getCountryCode()**

Description:

Gets the country code of the country the user is currently using. Country code is determined based on the user preferences set on the server. Country codes are those specified in ISO 3166.

Returns:

**String**

Returns a two-character country code as String.

Example:

```
function CSForm_OnOpen()
{
```

---

```
var usr = CSServer.GetCurrentUser();
var msg = "UserID: " + usr.getUserID() + "\r\n" +
"LoginID: " + usr.getLoginID() + "\r\n" +
"DisplayName: " + usr.getDisplayName() + "\r\n" +
"EmailAddress: " + usr.getEmailAddress() + "\r\n" +
"FirstName: " + usr.getFirstName() + "\r\n" +
"MiddleName: " + usr.getMiddleName() + "\r\n" +
"LastName: " + usr.getLastName() + "\r\n" +
"LanguageCode: " + usr.getLanguageCode() + "\r\n" +
"CountryCode: " + usr.getCountryCode();

CSForm.setStatusMsg( msg );
}
```

### **String getUserId()**

Description:

Gets the user ID. In LDAP installations, this is the unique full path user ID such as "cn=jsmith, cn=recipients, ou=sales".

Returns:

**String**

Returns the user ID as String.

Example:

```
function CSForm_OnOpen()
{
    var usr = CSServer.GetCurrentUser();

    var msg = "UserID: " + usr.getUserID() + "\r\n" +
"LoginID: " + usr.getLoginID() + "\r\n" +
"DisplayName: " + usr.getDisplayName() + "\r\n" +
"EmailAddress: " + usr.getEmailAddress() + "\r\n" +
"FirstName: " + usr.getFirstName() + "\r\n" +
"MiddleName: " + usr.getMiddleName() + "\r\n" +
"LastName: " + usr.getLastName() + "\r\n" +
"LanguageCode: " + usr.getLanguageCode() + "\r\n" +
"CountryCode: " + usr.getCountryCode();

    CSForm.setStatusMsg( msg );
}
```

---

## **String getLoginID()**

Description:

Gets the user's login ID. The login ID may be unique on the server depending on the server settings. The login ID is the ID the user types in to log in to LiquidOffice.

Returns:

**String**

Returns the user's login ID.

Example:

```
function CSForm_OnOpen()
{
    var usr = CSServer.getCurrentUser();

    var msg = "UserID: "      + usr.getUserID()      + "\r\n" +
              "LoginID: "     + usr.getLoginID()     + "\r\n" +
              "DisplayName: "  + usr.getDisplayName()  + "\r\n" +
              "EmailAddress: " + usr.getEmailAddress() + "\r\n" +
              "FirstName: "   + usr.getFirstName()   + "\r\n" +
              "MiddleName: "  + usr.getMiddleName()  + "\r\n" +
              "LastName: "    + usr.getLastname()    + "\r\n" +
              "LanguageCode: " + usr.getLanguageCode() + "\r\n" +
              "CountryCode: "  + usr.getCountryCode();

    CSForm.setStatusMsg( msg );
}
```

---

## **String getDisplayName()**

Description:

Gets the user's display name. The display name is typically first, middle, and last name together separated by a space though it can be different.

Returns:

**String**

Returns the user's display name.

Example:

```
function CSForm_OnOpen()
{
    var usr = CSServer.getCurrentUser();
```

---

```
var msg = "UserID: " + usr.getUserID() + "\r\n" +
    "LoginID: " + usr.getLoginID() + "\r\n" +
    "DisplayName: " + usr.getDisplayName() + "\r\n" +
    "EmailAddress: " + usr.getEmailAddress() + "\r\n" +
    "FirstName: " + usr.getFirstName() + "\r\n" +
    "MiddleName: " + usr.getMiddleName() + "\r\n" +
    "LastName: " + usr.getLastName() + "\r\n" +
    "LanguageCode: " + usr.getLanguageCode() + "\r\n" +
    "CountryCode: " + usr.getCountryCode();

    CSForm.setStatusMsg( msg );
}
```

---

### **String getEmailAddress()**

Description:

Gets the user's e-mail address.

Returns:

**String**

Returns the user's email address as String.

Example:

```
function CSForm_OnOpen()
{
    var usr = CSServer.getCurrentUser();

    var msg = "UserID: " + usr.getUserID() + "\r\n" +
        "LoginID: " + usr.getLoginID() + "\r\n" +
        "DisplayName: " + usr.getDisplayName() + "\r\n" +
        "EmailAddress: " + usr.getEmailAddress() + "\r\n" +
        "FirstName: " + usr.getFirstName() + "\r\n" +
        "MiddleName: " + usr.getMiddleName() + "\r\n" +
        "LastName: " + usr.getLastName() + "\r\n" +
        "LanguageCode: " + usr.getLanguageCode() + "\r\n" +
        "CountryCode: " + usr.getCountryCode();

    CSForm.setStatusMsg( msg );
}
```

---

**String getFirstName()**

Description:

Gets the user's first name.

Returns:

**String**

Returns the user's first name as String.

Example:

```
function CSForm_OnOpen()
{
    var usr = CSServer.getCurrentUser();

    var msg = "UserID: " + usr.getUserID() + "\r\n" +
              "LoginID: " + usr.getLoginID() + "\r\n" +
              "DisplayName: " + usr.getDisplayName() + "\r\n" +
              "EmailAddress: " + usr.getEmailAddress() + "\r\n" +
              "FirstName: " + usr.getFirstName() + "\r\n" +
              "MiddleName: " + usr.getMiddleName() + "\r\n" +
              "LastName: " + usr.getLastName() + "\r\n" +
              "LanguageCode: " + usr.getLanguageCode() + "\r\n" +
              "CountryCode: " + usr.getCountryCode();

    CSForm.setStatusMsg( msg );
}
```

---

**String getMiddleName()**

Description:

Gets the user's middle name.

Returns:

**String**

Returns the user's middle name as String.

Example:

```
function CSForm_OnOpen()
{
    var usr = CSServer.getCurrentUser();

    var msg = "UserID: " + usr.getUserID() + "\r\n" +
```

---

```
"LoginID: " + usr.getLoginID() + "\r\n" +
"DisplayName: " + usr.getDisplayName() + "\r\n" +
"EmailAddress: " + usr.getEmailAddress() + "\r\n" +
"FirstName: " + usr.getFirstName() + "\r\n" +
"MiddleName: " + usr.getMiddleName() + "\r\n" +
"LastName: " + usr.getLastName() + "\r\n" +
"LanguageCode: " + usr.getLanguageCode() + "\r\n" +
"CountryCode: " + usr.getCountryCode();"

CSForm.setStatusMsg( msg );
}
```

---

### **String getLastName()**

Description:

Gets the user's last name.

Returns:

**String**

Returns the user's last name as String.

Example:

```
function CSForm_OnOpen()
{
    var usr = CSServer.getCurrentUser();

    var msg = "UserID: " + usr.getUserID() + "\r\n" +
        "LoginID: " + usr.getLoginID() + "\r\n" +
        "DisplayName: " + usr.getDisplayName() + "\r\n" +
        "EmailAddress: " + usr.getEmailAddress() + "\r\n" +
        "FirstName: " + usr.getFirstName() + "\r\n" +
        "MiddleName: " + usr.getMiddleName() + "\r\n" +
        "LastName: " + usr.getLastName() + "\r\n" +
        "LanguageCode: " + usr.getLanguageCode() + "\r\n" +
        "CountryCode: " + usr.getCountryCode();

    CSForm.setStatusMsg( msg );
}
```

---

## CSRoute Object Specific Methods (Server Side)

The methods listed below are used at the CSRoute object level to access routing properties such as who the form will be routed to, carbon copied to, routing notes, routing level, etc.

---

### **String getEventID()**

Description:

Gets the ID of the current routing event. This ID is unique to each step of each routed form.

Returns:

**String**

Returns a unique routing ID.

Example:

```
function CSForm_OnRouteBefore()
{
    var msg = "Routing Details..." + "\r\n" +
        "EventID: "      + CSRoute.getEventID() + "\r\n" +
        "RouteTo: "       + CSRoute.getRouteTo() + "\r\n" +
        "CCTo: "         + CSRoute.getCCto() + "\r\n" +
        "LastNotes: "     + CSRoute.getNotes() + "\r\n" +
        "Status: "        + CSRoute.getStatus() + "\r\n" +
        "Level: "         + CSRoute.getLevel() + "\r\n" +
        "RoutingComplete: " + CSRoute.getRoutingComplete();

    CSServer.log("Before Route Event" + "\r\n" +
        msg );
}
```

---

### **String getRouteTo()**

Description:

Gets the user id, e-mail address, or Work Queue the form is being routed to. Available in OnRoutePage and OnRoute entry points.

Returns:

**String**

Returns the UserID, e-mail address, or Work Queue the form is being routed to or a blank String if there is no routing assignment.

---

Example:

```
function CSForm_OnRouteBefore()
{
    var msg = "Routing Details..." + "\r\n" +
        "EventID: " + CSRoute.getEventID() + "\r\n" +
        "RouteTo: " + CSRoute.getRouteTo() + "\r\n" +
        "CCTo: " + CSRoute.getCCto() + "\r\n" +
        "LastNotes: " + CSRoute.getNotes() + "\r\n" +
        "Status: " + CSRoute.getStatus() + "\r\n" +
        "Level: " + CSRoute.getLevel() + "\r\n" +
        "RoutingComplete: " + CSRoute.getRoutingComplete();
    CSServer.log("Before Route Event" + "\r\n" +
        msg );
}
```

---

### ***setRouteTo( String dest, String type )***

Description:

Sets the user id, e-mail address, or Work Queue to which the form should be routed. Available in CSForm\_OnRoutePage and CSForm\_OnRoute. When called in CSForm\_OnRoutePage, sets the default route to destination. When called in CSForm\_OnRoute, overrides any user routing destination except routing complete. Must call setRoutingComplete( false ) also to override routing complete. This function is ignored during a Reject operation.

Arguments:

**String - dest**

UserID or loginID address where the form will be routed.

**String type**

Optional, type of the destination specified by dest. One of, "User", "E-mail", or "Queue". If this parameter is not used dest will be interpreted as a userID or e-mail address. When an e-mail address is used, the system will try to resolve the address to a LiquidOffice User Account and send the form to that user's Inbox.

Example:

```
function CSForm_OnRoutePage()
{
    CSRoute.setRouteTo("jdoe@company.com", "E-mail");
    CSRoute.setShowPage(false);
}
```

---

### ***String getCCto()***

Description:

Gets a list of user id's to which the form will be CC'd. May be called in OnRoutePage, OnRoute or OnExport.

---

Returns:

**String**

Returns a comma separated list of user id's to which the form will be CC'd. Returns a blank String if the CC list is empty.

Example:

```
function CSForm_OnRouteBefore()
{
    var msg = "Routing Details..." + "\r\n" +
        "EventID: " + CSRoute.getEventID() + "\r\n" +
        "RouteTo: " + CSRoute.getRouteTo() + "\r\n" +
        "CCTo: " + CSRoute.getCCto() + "\r\n" +
        "LastNotes: " + CSRoute.getNotes() + "\r\n" +
        "Status: " + CSRoute.getStatus() + "\r\n" +
        "Level: " + CSRoute.getLevel() + "\r\n" +
        "RoutingComplete: " + CSRoute.getRoutingComplete();

    CSServer.log("Before Route Event" + "\r\n" +
        msg );
}
```

---

**addCCto( String userID )**

Description:

Appends a user id to the end of the CC list. May be called in OnRoutePage or OnRoute.

Arguments:

**String - userID**

UserID or loginID to add to the CC list.

Example:

```
function CSForm_OnRoute()
{
    CSRoute.addCCto("jdoe");
}
```

---

**String getNotes()**

Description:

Gets the routing notes. In OnRoutePage, getNotes() returns a blank String (no new notes have been entered yet). In OnRoute, getNotes() returns the notes just entered by the user or if none were entered, a blank String. getNotes cannot be called in any entry points besides OnRoutePage and OnRoute.

---

Returns:

**String**

Returns Notes as a String.

Example:

```
function CSForm_OnRouteBefore()
{
    var msg = "Routing Details..." + "\r\n" +
        "EventID: " + CSRoute.getEventID() + "\r\n" +
        "RouteTo: " + CSRoute.getRouteTo() + "\r\n" +
        "CCTo: " + CSRoute.getCCto() + "\r\n" +
        "LastNotes: " + CSRoute.getNotes() + "\r\n" +
        "Status: " + CSRoute.getStatus() + "\r\n" +
        "Level: " + CSRoute.getLevel() + "\r\n" +
        "RoutingComplete: " + CSRoute.getRoutingComplete();

    CSServer.log("Before Route Event" + "\r\n" +
        msg );
}
```

---

**Boolean setNotes( String notes )**

Description:

Sets a new routing note for the form. In OnRoutePage, this note pre-populates the notes field in the routing page. In OnRoute, calling setNotes, replaces any notes entered by the user with 'notes'. Calls to setNotes will be ignored in any entry points besides OnRoutePage and OnRoute.

Arguments:

**String - notes**

New routing notes as String.

Returns:

**Boolean**

Returns true on success and false on failure.

Example:

```
function CSForm_OnRoute()
{
    if( !CSRoute.getNotes() )
        CSRoute.setNotes("User added no notes.");
}
```

---

**String getStatus()**

Description:

Gets the routing status. Can be called from OnSubmit, OnValidateLookup, OnRoutePage, and OnRoute entry points. If the status is "Reject", or "Redirect" the CSForm object members are not defined.

Returns:

**String**

Returns "Approve", "Submit", "Reject", "Transfer", or "Redirect".

"Submit" Occurs when the user originates a routed form and submits it for the first time.

"Approve" Occurs when the user approves a routed form by selecting the "Approve" selection on the Submit Action button of the form.

"Reject" Occurs when the user rejects the form via the Submit Action on the form or via the Inbox.

"Transfer" Occurs when the user transfers the form via the Submit Action on the form or via the Inbox.

"Redirect" Occurs when the user redirects the form from the In Process folder.

If the status is "Reject" or "Transfer" or "Redirect" the CSForm object members are not defined for this entry point.

Example:

```
function CSForm_OnRouteBefore()
{
    var msg = "Routing Details..." + "\r\n" +
        "EventID: "      + CSRoute.getEventID() + "\r\n" +
        "RouteTo: "       + CSRoute.getRouteTo() + "\r\n" +
        "CCTo: "         + CSRoute.getCCto() + "\r\n" +
        "LastNotes: "     + CSRoute.getNotes() + "\r\n" +
        "Status: "        + CSRoute.getStatus() + "\r\n" +
        "Level: "         + CSRoute.getLevel() + "\r\n" +
        "RoutingComplete: " + CSRoute.getRoutingComplete();

    CSServer.log("Before Route Event" + "\r\n" +
        msg );
}
```

---

**Number getLevel()**

Description:

Gets the level (0 to n) of the routing progress. When a form is originally submitted, it's level is 0. When the first approver submits the form, it's level is 1, etc. If the form is rejected all the way back to the originator, it's level returns to 0. This function is defined during OnSubmit, OnRoutePage and OnRoute.

---

Returns:

**Number**

Returns the routing level.

Example:

```
function CSForm_OnRouteBefore()
{
    var msg = "Routing Details..." + "\r\n" +
        "EventID: " + CSRoute.getEventID() + "\r\n" +
        "RouteTo: " + CSRoute.getRouteTo() + "\r\n" +
        "CCTo: " + CSRoute.getCCto() + "\r\n" +
        "LastNotes: " + CSRoute.getNotes() + "\r\n" +
        "Status: " + CSRoute.getStatus() + "\r\n" +
        "Level: " + CSRoute.getLevel() + "\r\n" +
        "RoutingComplete: " + CSRoute.getRoutingComplete();
    CSServer.log("Before Route Event" + "\r\n" +
        msg );
}
```

---

**Boolean getRoutingComplete()**

Description:

Gets the state of the 'Routing Completed' radio button on the routing page. If a user has selected this option or a script function (setRoutingComplete) has set it, then this function will return true.

Returns:

**Boolean**

True if Routing Completed option is set. False otherwise.

Example:

```
function CSForm_OnRouteBefore()
{
    var msg = "Routing Details..." + "\r\n" +
        "EventID: " + CSRoute.getEventID() + "\r\n" +
        "RouteTo: " + CSRoute.getRouteTo() + "\r\n" +
        "CCTo: " + CSRoute.getCCto() + "\r\n" +
        "LastNotes: " + CSRoute.getNotes() + "\r\n" +
        "Status: " + CSRoute.getStatus() + "\r\n" +
        "Level: " + CSRoute.getLevel() + "\r\n" +
        "RoutingComplete: " + CSRoute.getRoutingComplete();
    CSServer.log("Before Route Event" + "\r\n" +
        msg );
}
```

---

### **setRoutingComplete( Boolean bComplete )**

Description:

Sets the routing complete flag. Available in OnRoutePage and OnRoute entry points. In OnRoutePage, this function specifies the default state for the routing complete radio button. In OnRoute, setting this flag overrides what the user selected in the routing page. This method has no effect except when called for Submit and Approve actions.

Arguments:

**Boolean - bComplete**

Pass true to complete routing or false to continue routing.

Example:

```
function CSForm_OnRoutePage()
{
    if( CSServer.GetCurrentUser().get UserID() == "jdoe" ){
        CSRoute.setRoutingComplete(true);
        CSRoute.setShowPage(false);
    }
}
```

---

### **setShowPage( Boolean bShowPage )**

Description:

Determines whether the Routing Page will be shown to the user or not. This method is only meaningful in the OnRoutePage() entry point and is ignored if the user selects multiple forms and clicks on Reject.

Arguments:

**Boolean - bShowPage**

Pass true to show the Routing Page or false to bypass.

Example:

```
function CSForm_OnRoutePage()
{
    if( CSServer.GetCurrentUser().get UserID() == "jdoe" ){
        CSRoute.setRoutingComplete(true);
        CSRoute.setShowPage(false);
    }
}
```

---

### **setEnabledRouteToCtrl( Boolean bEnable )**

Description:

Determines whether the RouteTo field on the Routing Page is enabled or not. This method is only meaningful in the OnRoutePage() entry point. It is ignored if the action is Reject.

---

Arguments:

**Boolean - bEnable**

Pass true to enable the RouteTo control or false to disable it.

Example:

```
function CSForm_OnRoutePage()
{
    if( CSRoute.getLevel() == 0 ){
        CSRoute.setRouteTo("jdoe");
        CSRoute.setEnableRouteToCtrl(false);
    }
}
```

---

**setEnableCCToCtrl( Boolean bEnable )**

Description:

Determines whether the CC To field on the Routing Page is enabled or not. This method is only meaningful in the OnRoutePage() entry point.

Arguments:

**Boolean - bEnable**

Pass true to enable the CC To field or false to disable it.

Example:

```
function CSForm_OnRoutePage()
{
    if( CSRoute.getLevel() == 0 )
        CSRoute.setEnableCCToCtrl(false);
}
```

---

**setEnableSubjectCtrl( Boolean bEnable )**

Description:

Determines whether the Subject field on the Routing Page is enabled or not. This method is only meaningful in the CSForm\_OnRoutePage() entry point. It is ignored if the action is Reject. The subject field is enabled by default.

Arguments:

**Boolean bEnable**

Pass true to enable the Subject field or false to disable it.

---

Example:

```
function CSForm_OnRoutePage()
{
    CSRoute.setEnableSubjectCtrl( false );
}
```

---

### ***String getSubject()***

Description:

Returns the subject associated with the routing event.

Returns:

**String**

Returns the subject as a String.

Example:

```
function CSForm_OnRoute()
{
    var f = CSForm.getField("Subject");
    var subj = CSRoute.getSubject();
    var to = CSServer.getCurrentUser().getEmailAddress();
    var from = "LiquidOffice Server<loadmin@cardiff.com>";
    var body = "You have routed the form successfully.';

    CSMail.send( to, from, subj, body );
}
```

---

### ***setSubject( String subject )***

Description:

Sets the subject associated with the routing event. A common use of this method is to set the subject line equal to the value of a form field that uniquely identifies the routing event.

Arguments:

**String subject**

Subject to associate with the routing event.

Example:

```
function CSForm_OnRoutePage()
{
    var f = CSForm.getField("Subject");
```

---

```
// Set the Routing Subject to the value of the Subject Entry Field
if (f.getValue() != "")
{
    CSRoute.setSubject( f.getValue() );
}
}
```

---

### **String getRouteToType()**

Description:

Used to indicate what type of target is returned in getRouteTo().

Returns:

**String**

Returns “User”, “E-mail” or “Queue” depending on the routing target.

Example:

```
function CSForm_OnRoute()
{
    var to = CSServer.getCurrentUser().getEmailAddress();
    var from = "LiquidOffice Server<loadadmin@cardiff.com>";
    var subject = CSRoute.getSubject();
    var body = "Routing Event Informational Email:" + "\r\n" +
        "RouteTo: " + CSRoute.getRouteTo() + "\r\n" +
        "TargetType: " + CSRoute.getRouteToType() + "\r\n" +
        "SenderId: " + CSServer.getCurrentUser().getUserID() + "\r\n" +
        "EventID: " + CSRoute.getEventID() + "\r\n";
    CSMail.send( to, from, subject, body );
}
```

---

## Utility Objects (Server Side)

The LiquidOffice Server Script object model provides several utility functions for accessing database, file, and email operations. The global utility objects available are:

- CSFile
- CSMail
- CSDB

Additionally, the following objects can be accessed through these top level objects:

- CSFileReader
- CSFileWriter
- CSConnection
- CSRowSet
- CSColumn

In addition to the Server Utility objects supplied, it is also possible to import Java objects and use them to enhance script writing. Unlike the previously mentioned JavaScript objects, these Objects are imported Java Objects and therefore behave a little differently. Java uses a method called ‘.equals()’ to compare two objects rather than the familiar ‘==’ operator. In Java, ‘==’ is used to determine if the objects reference the same instance. Therefore objects returned from the utility objects should be compared in the following manner:

```
//  
// This is an example of a string comparison  
//  
var str = filereader.read();  
if (str.equals( "A" )) {  
    // do something  
}
```

Importing Java objects can be done in several ways. First, by accessing the fully qualified Java object name, and second, by using the importClass() and/or importPackage() JavaScript functions, then creating the Java object without the path. Here are examples:

```
//  
// Retrieve a random number using the fully qualified Java object name.  
//  
var random = new java.util.Random();  
var random_number = random.nextInt();
```

---

```
...
//  
//  Retrieve a random number by first importing the Java object, then creating the object.  
//  
importClass( java.util.Random ); // could also import the entire package like this . . . importPackage( java.util );  
var random = new Random();  
var random_number = random.nextInt();
```

**NOTE:** All imported Java objects must be in the Server's Java class path.

**NOTE:** Fully qualified Java object names that do not begin with "java" must be prefaced with "Packages" i.e. importing javax.rmi would be done like this: importPackage( Packages.java.rmi );.

---

## File Access

### CSFile Object Specific Methods (Server Side)

The CSFile object is used to provide a simple interface to the file system. The file names specified are relative to single designated directory on the server for form scripts. For export scripts the file name can be full path name to any file on the system.

---

#### ***CSFileReader newFileReader( String fileName )***

Description:

Creates File reader object with access to a common server subdirectory for form script or access to the full path file for export script. In form script, this also handles full path file names from the attachments object.

Arguments:

**String - fileName**

Pathname of the file.

Returns:

**CSFileReader**

Returns a CSFileReader object.

Example:

See CSFileReader Object Specific Methods Section.

---

### ***CSFileWriter newFileWriter( String fileName, Boolean bAppend )***

Description:

Creates File writer object with access to a common server subdirectory for form script or access to the full path file for export script.

Arguments:

**String - fileName**

Pathname of the file.

**Boolean - bAppend**

Pass true to cause an existing file to be appended to or false to overwrite an existing file. If true is passed but the file does not yet exist, a new file will be created.

Returns:

**CSFileWriter**

Returns a CSFileWriter object for the file.

Example:

See CSFileWriter Object Specific Methods Section.

---

### ***Boolean exists( String fileName )***

Description:

Indicates whether the file exists or not. Filename is relative to a common server subdirectory for form script and must be a full path for export script.

Arguments:

**String - fileName**

Pathname of the file.

Returns:

**Boolean**

Returns true if the file exists or false if not.

Example:

```
var bExist = CSFile.exists("TestFile.txt");
```

---

### ***Boolean deleteFile( String filepath )***

Description:

Deletes the file at the location specified by filepath. LiquidOffice Server must have proper permissions to delete the file.

Arguments:

**String filepath**

Path of file to delete.

---

Returns:

**Boolean**

Returns true on success or false on failure.

Example:

```
CSForm_OnExport()
{
    var filename = "TestFile.txt"; // File resides in Attachments directory
    var bSuccess = false;

    bSuccess = CSFile.deleteFile( filename );

    if (!bSuccess)
        CSServer.log("Unable to delete file!");
}
```

---

## CSFileReader Object Specific Methods (Server Side)

The CSFileReader object provides simple read access to the file system

---

### Number size()

Description:

Gets the file size in bytes.

Returns:

**Number**

Returns the file size in bytes.

Example:

```
function CSForm_OnOpen()
{
    var fileName = "TestFile.txt";
    var file = null; // FileReader object

    // Check to see if file exists
    if (!CSFile.exists( fileName )) {
        CSForm.setStatusMsg( "The " + fileName + " file does not exist!" );
        return;
    }

    // Open new FileReader (File Located in Attachments directory)
    file = CSFile.newFileReader( fileName );

    // Report File Size
    CSForm.getField("FileSize").setValue( file.size() );
}
```

---

### String read()

Description:

Reads one character from the file.

Returns:

**String**

Returns a single character or `null` if if the end of the file has been reached.

Example:

```
function CSForm_OnOpen()
{
    var custField = CSForm.getField("Customers");
```

---

```
var fileName = "CustomerList.txt";
var file = null; // FileReader object
var buffer = null; // FileReader buffer

// Check to see if file exists
if (!CSFile.exists( fileName )) {
    CSForm.setStatusMsg( "The " + fileName + " file does not exist!" );
    return;
}

// Open new FileReader (File Located in Attachments directory)
file = CSFile.newFileReader( fileName );

// Read File into Form Field
buff = file.read(); // Read first
while (buff) {
    custField.setValue( custField.getValue() + buff );
    buff = file.read(); // Read next
}

// Close the FileReader
file.close()
}
```

### **Number readByte()**

Description:

Reads one byte from the file.

Returns:

**Number**

Returns a single byte as an Number or -1 if the end of the file has been reached.

Example:

```
function CSForm_OnOpen()
{
    var custField = CSForm.getField("Customers");
    var fileName = "CustomerList.txt";
    var file = null; // FileReader object
    var buffer = null; // FileReader buffer
```

---

```
// Check to see if file exists
if (!CSFile.exists( fileName )) {
    CSForm.setStatusMsg( "The " + fileName + " file does not exist!" );
    return;
}

// Open new FileReader (File Located in Attachments directory)
file = CSFile.newFileReader( fileName );
// Read File into Form Field
buff = file.readByte(); // Read first
while (buff != -1) {
    custField.setValue( custField.getValue() + String.fromCharCode( buff ) );
    buff = file.readByte(); // Read next
}
// Close the FileReader
file.close()
}
```

---

### ***String* **readLine()****

Description:

Reads one line from the file. The read is performed from the current position in the file to the next newline character.

Returns:

**String**

Returns a single line as String or null if the end of the file has been reached.

Example:

```
function CSForm_OnOpen()
{
    var custField = CSForm.getField("Customers");
    var fileName = "CustomerList.txt";
    var file = null; // FileReader object
    var buffer = null; // FileReader buffer
    // Check to see if file exists
    if (!CSFile.exists( fileName )) {
        CSForm.setStatusMsg( "The " + fileName + " file does not exist!" );
        return;
    }
```

---

```
// Open new FileReader (File Located in Attachments directory)
file = CSFile.newFileReader( fileName );
// Read File into Form Field
buff = file.readLine(); // Read first
while (buff) {
    custField.setValue( custField.getValue() + buff + "\r\n" );
    buff = file.readLine(); // Read next
}
// Close the FileReader
file.close()
}
```

---

## **Number skip( Number n )**

Description:

Skips n bytes. Number of bytes actually skipped may be different than n if the end of the file is reached.

Arguments:

### **Number n**

Number of bytes to skip.

Returns:

### **Number**

Returns n.

Example:

```
function CSForm_OnOpen()
{
    var custField = CSForm.getField("Customers");
    var fileName = "CustomerList.txt";
    var file = null; // FileReader object
    var buffer = null; // FileReader buffer
    var size = null; // File size in bytes

    // Check to see if file exists
    if (!CSFile.exists( fileName )) {
        CSForm.setStatusMsg( "The " + fileName + " file does not exist!" );
        return;
    }

    // Open new FileReader (File Located in Attachments directory)
    file = CSFile.newFileReader( fileName );

    // Report File Size
    size = file.size();

    // Move to the half-point of the file
    file.skip(size / 2);

    // Read File into Form Field
    buff = file.readLine(); // Read first
    while (buff) {
        custField.setValue( custField.getValue() + buff + "\r\n" );
        buff = file.readLine(); // Read next
    }
}
```

```
    }
    // Close the FileReader
    file.close();
}
```

## **close()**

Description:

Closes the file. Once the file is closed, the CSFileReader object can no longer be used.

Returns:

**Boolean**

Returns true on success and false on failure.

Example:

```
function CSForm_OnOpen()
{
    var custField = CSForm.getField("Customers");
    var fileName = "CustomerList.txt";
    var file = null; // FileReader object
    var buffer = null; // FileReader buffer

    // Check to see if file exists
    if (!CSFile.exists( fileName )) {
        CSForm.setStatusMsg( "The " + fileName + " file does not exist!" );
        return;
    }

    // Open new FileReader (File Located in Attachments directory)
    file = CSFile.newFileReader( fileName );

    // Read File into Form Field
    buff = file.readLine(); // Read first
    while (buff) {
        custField.setValue( custField.getValue() + buff + "\r\n" );
        buff = file.readLine(); // Read next
    }
    // Close the FileReader
    file.close();
}
```

---

## CSFileWriter Object Specific Methods (Server Side)

The CSFileWriter object provides simple write access to the file system

---

### **Boolean write( String str )**

Description:

Writes a string to the file.

Arguments:

**String str**

String to write to the file.

Returns:

**Boolean**

Returns true on success and false on failure.

Example:

```
function CSForm_OnSubmit()
{
    var newCustField = CSForm.getField("NewCustomerName");
    var fileName = "CustomerList.txt";
    var file = null; // FileWriter object

    // Check to see if file exists
    if (!CSFile.exists( fileName )) {
        CSForm.setStatusMsg( "The " + fileName + " file does not exist!" );
        return;
    }

    // Open new FileWriter in append mode (File Located in Attachments directory)
    file = CSFile.newFileWriter( fileName, true );

    // Write the New Customer Name to the end of the file
    file.write( newCustField.getValue() + "\r\n" );

    // Close the FileWriter
    file.close();
}
```

---

***Boolean writeByte( Number n )***

Description:

Writes a single byte to the file.

Arguments:

**Number n**

Byte to write to the file.

Returns:

**Boolean**

Returns true on success or false on failure.

Example:

```
function CSForm_OnComplete()
{
    /**
     ** Create Backup Copy File
     **/

    var fileName = "CustomerList.txt";
    var bakupName = "CustomerList.bak";
    var srcFile = null; // FileReader object
    var bakFile = null; // FileWriter object
    var buff = null; // Read Buffer

    // Check to see if file exists
    if (!CSFile.exists( fileName )) {
        CSForm.setStatusMsg( "The " + fileName + " file does not exist!" );
        return;
    }

    // Open new FileReader (File Located in Attachments directory)
    srcFile = CSFile.newFileReader( fileName );

    // Open new FileWriter in overwrite mode (File will be in Attachments directory)
    bakFile = CSFile.newFileWriter( bakupName );

    // Loop through original file one byte at a time
    buff = srcFile.readByte();
    while (buff != -1) {
        bakFile.writeByte( buff );
    }
}
```

---

```
buff = srcFile.readByte();
}

// Compare files
if (srcFile.size() != bakFile.size()) {
    CSServer.log("Problem creating backup file!!");
}

// Close files
srcFile.close();
bakFile.close();
}
```

---

### ***Boolean writeLine( String str )***

Description:

Writes a String and appends carriage return and line feed.

Arguments:

**String str**

String to write to the file.

Returns:

**Boolean**

Returns true on success and false on failure.

Example:

```
function CSForm_OnSubmit()
{
    var newCustField = CSForm.getField("NewCustomerName");
    var fileName = "CustomerList.txt";
    var file = null; // FileWriter object

    // Check to see if file exists
    if (!CSFile.exists( fileName )) {
        CSForm.setStatusMsg( "The " + fileName + " file does not exist!" );
        return;
    }

    // Open new FileWriter in append mode (File Located in Attachments directory)
    file = CSFile.newFileWriter( fileName, true );
```

---

```
// Write the New Customer Name to the end of the file
file.writeLine( newCustField.getValue() );
// Close the FileWriter
file.close();
}
```

## **close()**

Description:

Closes the file. Once a file is closed, the CSFileWriter object cannot be used. It is good practice to always close files when done using them. It is required that files be closed before opening them again.

LiquidOffice Server will close any CSFileReader or CSFileWriter objects left open after a script exits.

Returns:

**Boolean**

Returns true on success and false on failure.

Example:

```
function CSForm_OnOpen()
{
    var custField = CSForm.getField("Customers");
    var fileName = "CustomerList.txt";
    var file = null; // FileReader object
    var buffer = null; // FileReader buffer
    // Check to see if file exists
    if (!CSFile.exists( fileName )) {
        CSForm.setStatusMsg( "The " + fileName + " file does not exist!" );
        return;
    }
    // Open new FileReader (File Located in Attachments directory)
    file = CSFile.newFileReader( fileName );
    // Read File into Form Field
    buff = file.readLine(); // Read first
    while (buff) {
        custField.setValue( custField.getValue() + buff + "\r\n" );
        buff = file.readLine(); // Read next
    }
    // Close the FileReader
    file.close();
}
```

---

## Email Access

### CSEmail Object Specific Methods (Server Side)

The CSEmail object provides simple access to email.

---

#### **Boolean send( String to, String from, String subject, String body )**

Description:

Sends e-mail to the specified address with subject-line and text-body.

Arguments:

**String - to**

Address the e-mail will be sent to.

**String - from**

Address that will appear in the From line. If "from" is missing, the e-mail address set in Server Settings for system administrators shall be used.

**String - subject**

E-mail subject.

**String - body**

E-mail body.

Returns:

**Boolean**

Returns true on success and false on failure.

Example:

```
function CSForm_OnExport()
{
    var email = CSForm.getField("Email").getValue();
    var confirm = CSForm.getField("OrderConfirm").getValue();
    var sender = "orders@mbw.com";
    var subject = "Order Receipt Confirmation From MBW.com";
    var shipBill = CSForm.getField("SameShippingCheck").getValue();
    var shipMthd = CSForm.getField("Shipping_Method").getValue();

    var body = "ORDER DETAILS" +
        "\r\nShip To:" +
        "\r\n" + CSForm.getField("Ship_To_Company").getValue() +
        "\r\n" + CSForm.getField("Ship_To_ATTN").getValue() +
        "\r\n" + CSForm.getField("Ship_To_Address").getValue() +
```

---

```

"\r\t" + CSForm.getField("Ship_To_City").getValue() +
", " + CSForm.getField("Ship_To_State").getValue() +
" " + CSForm.getField("Ship_To_Zip").getValue() +
"\r\rBill To:";

if ( shipBill == "0" )
{
    body += "\r\t" + CSForm.getField("Bill_To_Company").getValue() +
    "\r\t" + CSForm.getField("Bill_To_ATTN").getValue() +
    "\r\t" + CSForm.getField("Bill_To_Address").getValue() +
    "\r\t" + CSForm.getField("Bill_To_City").getValue() +
    ", " + CSForm.getField("Bill_To_State").getValue() +
    " " + CSForm.getField("Bill_To_Zip").getValue();
}

else
{
    body += "\r\t--Same as Ship To Address--";
}

// Send Confirmation Email
if ( email != "" && email != null && confirm == "1" )
{
    if ( CSMail.send( email, sender, subject, body ) )
        CSServer.log("Confirmation email sent successfully.");
    else
        CSServer.log("Error sending email confirmation!!!");
}

```

---

### **Boolean sendWithAttach( String to, String from, String subject, String body, Array attachPaths )**

Description:

Sends e-mail to the specified address with subject-line and text-body. Also attaches files specified by attachPaths array.

Arguments:

**String to**

Address the e-mail will be sent to.

**String from**

Address that will appear in the From line. If “from” is missing, the e-mail address set in Server Settings for system administrators shall be used.

---

**String subject**

E-mail subject.

**String body**

E-mail body.

**Array attachPaths**

Array of file attachment path Strings.

Returns:

**Boolean**

Returns true on success and false on failure.

Example:

```
function CSForm_OnExport()
{
    var to      = CSForm.getField("To").getValue();
    var from    = CSForm.getField("From").getValue();
    var cc      = CSForm.getField("CC").getValue();
    var subject = CSForm.getField("Subject").getValue();
    var message = CSForm.getField("Message").getValue();
    var attachments = CSForm.getAttachments();

    if (attachments && attachments.length > 0)
        CSMail.sendWithAttach( to, from, subject, message, attachments );
    else
        CSMail.send( to, from, subject, message );
}
```

---

## Database Access

### CSDB Object Specific Methods (Server Side)

The CSDB object provides a simple interface to databases.

---

#### **array getDataLinkList()**

Description:

List of names of all Connect Agents defined on the server of SQL type.

Returns:

**Array**

Returns a String array listing available Connect Agents or a zero length array if no Connect Agents are defined. Returns null if an error occurred.

Example:

```
function CSForm_OnOpen()
{
    /**
     ** Write DataLink List to Server Log
     */
    var msg = "Listing Server Connect Agents" + "\r\n";
    var aLinks = CSDB.getDataLinkList();

    if (aLinks.length > 0) {
        for (var i = 0; i < aLinks.length; i++)
            msg += i + " - " + aLinks[i] + "\r\n";
        CSServer.log(msg);
    }
}
```

---

#### **CSConnection newConnection( String dataLinkName )**

Description:

Creates a new CSConnection object for the Connect Agent name specified.

Arguments:

**String - dataLinkName**

Connect Agent name.

Returns:

**CSConnection**

Returns a new CSConnection object or null if the connection could not be created.

---

Example:

```
function CSForm_OnOpen()
{
    var Tables = null; // Array for table list
    var con = null;   // DB Connection Object
    var dLink = "PERF_TEST"; // Connect Agent Name
    var msg = ""; // String Buffer

    // Add first line of log entry
    msg = "Tables List From " + dLink + " Connect Agent:" + "\r\n";

    // Open New Connection
    con = CSDB.newConnection( dLink );
    if (!con) {
        CSForm.setStatusMsg( "Unable to open database connection!" );
        return;
    }

    // Retrieve Tables List
    Tables = con.getTableList();

    // Add each table name to string buffer
    for (var i = 0; i < Tables.length; i++)
        msg += Tables[i] + "\r\n";

    // Write String Buffer to Server Log
    CSServer.log(msg);

    // Close Connection
    con.close();
}
```

---

#### ***CSConnection newExternalConnection( String URL, String driver, String user, String password )***

Description:

Creates a new CSConnection object for the specified URL, driver, user, and password. Will attempt JNDI connection if JDBC cannot be opened.

---

Arguments:

**String - URL**

The URL of the database to connect to.

**String - driver**

Name of the driver to be used.

**String - user**

Username to connect with.

**String - password**

Password for the username specified.

Returns:

**CSConnection**

Returns a new CSConnection object or null if the connection could not be created.

Example:

```
function CSForm_OnOpen()
{
    var Tables = null; // Array for table list
    var con = null; // DB Connection Object
    var dURL = "jdbc:oracle:thin:@192.168.0.104:1521:V2ITEST"; // Database URL
    var dDriver = "oracle.jdbc.driver.OracleDriver"; // Database Driver
    var dUser = "scott"; // Database user
    var dPass = "tiger"; // Database password
    var msg = ""; // String Buffer

    // Add first line of log entry
    msg = "Tables List From " + dURL + "\r\n";

    // Open New Connection
    con = CSDB.newExternalConnection( dURL, dDriver, dUser, dPass );
    if (!con) {
        CSForm.setStatusMsg( "Unable to open database connection!" );
        return;
    }

    // Retrieve Tables List
    Tables = con.getTableList();
    CSServer.log(Tables.length);
    // Add each table name to string buffer
```

---

```
for (var i = 0; i < Tables.length; i++)
msg += Tables[i] + "\r\n";

// Write String Buffer to Server Log
CSServer.log(msg);

// Close Connection
con.close();
}
```

## CSConnection Object Specific Methods (Server Side)

The CSConnection object provides access to database table lists, tables' column lists, and the ability to run SQL queries. Use CSDB to create CSConnection objects.

---

### **array getTableList()**

Description:

List all table names in the database referenced by the connection.

Returns:

#### **Array**

Returns an array of String table names in the database or a 0 length array if the database contains no tables, or null if an error occurred.

Example:

```
function CSForm_OnOpen()
{
    var Tables = null; // Array for table list
    var con = null; // DB Connection Object
    var dLink = "PERF_TEST"; // Connect Agent Name
    var msg = ""; // String Buffer

    // Add first line of log entry
    msg = "Tables List From " + dLink + " Connect Agent:" + "\r\n";

    // Open New Connection
    con = CSDB.newConnection( dLink );
    if (!con) {
        CSForm.setStatusMsg( "Unable to open database connection!" );
        return;
    }
}
```

---

```
}

// Retrieve Tables List
Tables = con.getTableList();

// Add each table name to string buffer
for (var i = 0; i < Tables.length; i++)
    msg += Tables[i] + "\r\n";

// Write String Buffer to Server Log
CSServer.log(msg);

// Close Connection
con.close();
}
```

---

**array getColumnList( String tableName )**

Description:

Gets an array of CSColumn objects for each column in the specified table.

Arguments:

**String - tableName**

Name of the table.

Returns:

**Array**

Returns an array of CSColumn objects for each column in the specified table or null if an error occurred.

Example:

```
function CSForm_OnOpen()
{
    var Columns = null; // Array for column list
    var con = null; // DB Connection Object
    var dLink = "PERF_TEST"; // Connect Agent Name
    var table = "1040Dependants"; // Table Name
    var msg = ""; // String Buffer

    // Add first line of log entry
    msg = "Columns List From " + dLink + ", " + table + ": \r\n";
```

---

```
// Open New Connection
con = CSDB.newConnection( dLink );
if (!con) {
    CSForm.setStatusMsg( "Unable to open database connection!" );
    return;
}

// Retrieve Columns List
Columns = con.getColumnList( table );

// Add each column name and type to string buffer
for (var i = 0; i < Columns.length; i++)
    msg += "Name: " + Columns[i].getName() + " - Type: " + Columns[i].getType() + "\r\n";

// Write String Buffer to Server Log
CSServer.log(msg);

// Close Connection
con.close();
}
```

---

**Boolean close()**

Description:

Closes the connection. Once the connection is closed, the CSConnection object and any derived row or column objects cannot be used.

Returns:

**Boolean**

Returns true on success and false on failure.

Example:

```
function CSForm_OnOpen()
{
    var Tables = null; // Array for table list
    var con = null; // DB Connection Object
    var dLink = "PERF_TEST"; // Connect Agent Name
    var msg = ""; // String Buffer
    // Add first line of log entry
    msg = "Tables List From " + dLink + " Connect Agent:" + "\r\n";

    // Open New Connection
    con = CSDB.newConnection( dLink );
    if (!con) {
        CSForm.setStatusMsg( "Unable to open database connection!" );
        return;
    }

    // Retrieve Tables List
    Tables = con.getTableList();

    // Add each table name to string buffer
    for (var i = 0; i < Tables.length; i++)
        msg += Tables[i] + "\r\n";

    // Write String Buffer to Server Log
    CSServer.log(msg);

    // Close Connection
    con.close();
}
```

---

### **Number executeUpdate( String SQL )**

Description:

Executes a SQL command that does not return rows such as CREATE, DELETE, UPDATE or INSERT.

Arguments:

**String - SQL**

SQL command to execute.

Returns:

**Number**

Returns the number of rows updated.

Example:

```
function CSForm_OnOpen()
{
    var con = null; // DB Connection Object
    var dLink = "PERF_EXPORT"; // Connect Agent Name
    var table = "1040A_Data"; // Table Name
    var sql = "DROP TABLE \"" + table + "\""; // SQL Statement to execute
    var r = null;

    // Open New Connection
    con = CSDB.newConnection( dLink );
    if (!con) {
        CSForm.setStatusMsg( "Unable to open database connection!" );
        return;
    }

    // Execute Update SQL Statement to delete table
    r = con.executeUpdate( sql );

    // Display message indicating number of rows affected by update
    CSForm.setStatusMsg( r + " rows affected." );

    // Close Connection
    con.close();
}
```

---

## ***CSRowSet executeSQL( String SQL )***

Description:

Executes a SQL command that returns rows.

Arguments:

**String - SQL**

SQL command to execute.

Returns:

**CSRowSet**

Returns a CSRowSet object if the SQL query is executed successfully or null on failure.

Example:

```
function CSForm_OnOpen()
{
    var Columns = null; // Array for column list
    var con = null; // DB Connection Object
    var dLink = "PERF_TEST"; // Connect Agent Name
    var table = "1040Dependants"; // Table Name
    var sql = "SELECT * FROM \"" + table + """; // SQL Statement to execute
    var rows = null; // Rowset Object Variable
    var dbList = CSForm.getField("dbList").getChoices();
    var cols = 0;

    // Open New Connection
    con = CSDB.newConnection( dLink );
    if (!con) {
        CSForm.setStatusMsg( "Unable to open database connection!" );
        return;
    }

    // Open Rowset
    rows = con.executeSQL( sql );
    if (!rows) {
        CSForm.setStatusMsg( "Unable to open rowset!" );
        return;
    }

    // Remember Column Count
    cols = rows.getColumnCount();
```

---

```
// Add each row to the list field
while (rows.nextRow()) {
    var val = "";
    for (var i = 1; i <= cols; i++)
        val += rows.getValue(i) + ",";
    dbList.insertAt( -1, val, val );
}

// Close Rowset
rows.close();

// Close Connection
con.close();
}
```

## CSRowSet Object Specific Methods (Server Side)

The CSRowSet object provides access to a row's individual columns and data. Use CSConnection to create CSRowSet objects.

---

### **Number getColumnCount()**

Description:

Gets the number of columns in the rowset.

Returns:

#### **Number**

Returns the number of columns in the rowset.

Example:

```
function CSForm_OnOpen()
{
    var Columns = null; // Array for column list
    var con = null; // DB Connection Object
    var dLink = "PERF_TEST"; // Connect Agent Name
    var table = "1040Dependants"; // Table Name
    var sql = "SELECT * FROM \\" + table + "\\"; // SQL Statement to execute
    var rows = null; // Rowset Object Variable
    var dbList = CSForm.getField("dbList").getChoices();
```

---

```
var cols = 0;

// Open New Connection
con = CSDB.newConnection( dLink );
if (!con) {
    CSForm.setStatusMsg( "Unable to open database connection!" );
    return;
}

// Open Rowset
rows = con.executeSQL( sql );
if (!rows) {
    CSForm.setStatusMsg( "Unable to open rowset!" );
    return;
}

// Remember Column Count
cols = rows.getColumnCount();

// Add each row to the list field
while (rows.nextRow()) {
    var val = "";
    for (var i = 1; i <= cols; i++)
        val += rows.getValue(i) + ",";
    dbList.insertAt( -1, val, val );
}

// Close Rowset
rows.close();

// Close Connection
con.close();
}
```

---

***array getColumnList()***

Description:

Gets a CSCColumn object for each column in the rowset in an array.

Returns:

**Array**

Returns an array of CSCColumn objects or a 0 length array if there are no columns in the rowset, or null if an error occurred.

Example:

```
function CSForm_OnOpen()
{
    var Columns = null; // Array for column list
    var con = null; // DB Connection Object
    var dLink = "PERF_TEST"; // Connect Agent Name
    var table = "1040Dependants"; // Table Name
    var sql = "SELECT * FROM \\" + table + "\\"; // SQL Statement to execute
    var rows = null; // Rowset Object Variable
    var dbList = CSForm.getField("dbList").getChoices();
    var cols = null; // Column Object Array
    var buff = ""; // String buffer

    // Open New Connection
    con = CSDB.newConnection( dLink );
    if (!con) {
        CSForm.setStatusMsg( "Unable to open database connection!" );
        return;
    }

    // Open Rowset
    rows = con.executeSQL( sql );
    if (!rows) {
        CSForm.setStatusMsg( "Unable to open rowset!" );
        return;
    }

    // Retrieve Columns
    cols = rows.getColumnList();
    if (!cols) {
```

---

```
    CSForm.setStatusMsg( "Unable to retrieve columns!" );
    return;
}

// Add each column's info to the list field
for (var i = 0; i < rows.getColumnCount(); i++) {
    buff = "Name: " + cols[i].getName() +
        " - Type: " + cols[i].getType() +
        " - Size: " + cols[i].getSize() +
        " - Nullable: " + cols[i].isNullable();
    dbList.insertAt( i, i, buff );
}

// Close Rowset
rows.close();

// Close Connection
con.close();
}
```

---

### ***String getColumnName( Number n )***

Description:

Gets the name of the n-th column.

Arguments:

**Number - n**

Index of the column. Column indices are one based. The first column has an index of 1.

Returns:

**String**

Returns the column name.

Example:

```
function CSForm_OnOpen()
{
    /**
     ** Set Field Value equal to 3rd column's name
     */
    var con = null; // DB Connection Object
```

---

```
var dLink = "PERF_TEST"; // Connect Agent Name
var table = "1040Dependants"; // Table Name
var sql = "SELECT * FROM \"" + table + "\""; // SQL Statement to execute
var rows = null; // Rowset Object Variable

// Open New Connection
con = CSDB.newConnection( dLink );
if (!con) {
    CSForm.setStatusMsg( "Unable to open database connection!" );
    return;
}

// Open Rowset
rows = con.executeSQL( sql );
if (!rows) {
    CSForm.setStatusMsg( "Unable to open rowset!" );
    return;
}

// Set Value of Entry1 field to 3rd column's name
CSForm.getField("Entry1").setValue( rows.getColumnName(3) );

// Close Rowset
rows.close();

// Close Connection
con.close();
}
```

---

#### ***Number getColumnIndex( String columnName )***

Description:

Gets the index of the column.

Arguments:

**String - columnName**

Name of the column.

---

Returns:

### Number

Returns the index of the column. Column indices are one based. The first column will have an index of 1.

Example:

```
function CSForm_OnOpen()
{
    /**
     ** Set Field Value equal to DependantName column's index
     **/

    var con = null; // DB Connection Object
    var dLink = "PERF_TEST"; // Connect Agent Name
    var table = "1040Dependants"; // Table Name
    var sql = "SELECT * FROM \\" + table + "\\"; // SQL Statement to execute
    var rows = null; // Rowset Object Variable

    // Open New Connection
    con = CSDB.newConnection( dLink );
    if (!con) {
        CSForm.setStatusMsg( "Unable to open database connection!" );
        return;
    }

    // Open Rowset
    rows = con.executeSQL( sql );
    if (!rows) {
        CSForm.setStatusMsg( "Unable to open rowset!" );
        return;
    }

    // Set Value of Entry1 field to DependantName column's index
    CSForm.getField("Entry1").setValue( rows.getColumnIndex("DependantName") );

    // Close Rowset
    rows.close();

    // Close Connection
    con.close();
}
```

---

### **String getColumnType( Number n )**

Description:

Gets the data type of the n-th column.

Arguments:

**Number - n**

Index of the column. Column indices are one based. The first column will have an index of 1.

Returns:

**String**

Returns the data type of the column as String. Typical return values are "String", "Numeric", "Binary", "Date", "Time" or "TimeStamp", however some databases may return "varchar", "Number", "bin", "date", etc. These will vary with the database you use.

Example:

```
function CSForm_OnOpen()
{
    /**
     * Set Field Value equal to 3rd column's data type
     */

    var con = null; // DB Connection Object
    var dLink = "PERF_TEST"; // Connect Agent Name
    var table = "1040Dependants"; // Table Name
    var sql = "SELECT * FROM '" + table + "'"; // SQL Statement to execute
    var rows = null; // Rowset Object Variable

    // Open New Connection
    con = CSDB.newConnection( dLink );
    if (!con) {
        CSForm.setStatusMsg( "Unable to open database connection!" );
        return;
    }

    // Open Rowset
    rows = con.executeSQL( sql );
    if (!rows) {
        CSForm.setStatusMsg( "Unable to open rowset!" );
        return;
    }
}
```

---

```
}

// Set Value of Entry1 field to 3rd column's data type
CSForm.getField("Entry1").setValue( rows.getColumnType(3) );

// Close Rowset
rows.close();

// Close Connection
con.close();
}
```

---

### ***String getValue( Number n )***

Description:

Gets the value of the n-th column in the current row.

Arguments:

**Number - n**

Index of the column. Column indices are one based. The index of the first column is 1.

Returns:

**String**

Returns the value of the column in the current row or null on failure or if n is not in the range 1 to getColumnCount().

Example:

```
function CSForm_OnOpen()
{
    /**
     * Set Field Value equal to 3rd column's value
     */
    var con = null; // DB Connection Object
    var dLink = "PERF_TEST"; // Connect Agent Name
    var table = "1040Dependants"; // Table Name
    var sql = "SELECT * FROM \\" + table + "\\"; // SQL Statement to execute
    var rows = null; // Rowset Object Variable

    // Open New Connection
    con = CSDB.newConnection( dLink );
    if (!con) {
```

---

```
CSForm.setStatusMsg( "Unable to open database connection!" );
return;
}

// Open Rowset
rows = con.executeSQL( sql );
if (!rows) {
    CSForm.setStatusMsg( "Unable to open rowset!" );
    return;
}

// Set Value of Entry1 field to 3rd column's value
CSForm.getField("Entry1").setValue( rows.getValue(3) );

// Close Rowset
rows.close();

// Close Connection
con.close();
}
```

---

***String getValue( String columnName )***

Description:

Gets the value of the specified column in the current row.

Arguments:

**String - columnName**

Name of the column.

Returns:

**String**

Returns the value of the named column in the current row or null on failure or if columnName is not a valid column name in the rowset.

Example:

```
function CSForm_OnOpen()
{
    /**
     ** Set Field Value equal to DependantName column's value
     */
    var con = null; // DB Connection Object
    var dLink = "PERF_TEST"; // Connect Agent Name
    var table = "1040Dependants"; // Table Name
    var sql = "SELECT * FROM \\" + table + "\\"; // SQL Statement to execute
    var rows = null; // Rowset Object Variable

    // Open New Connection
    con = CSDB.newConnection( dLink );
    if (!con) {
        CSForm.setStatusMsg( "Unable to open database connection!" );
        return;
    }

    // Open Rowset
    rows = con.executeSQL( sql );
    if (!rows) {
        CSForm.setStatusMsg( "Unable to open rowset!" );
        return;
    }

    // Set Value of Entry1 field to DependantName column's value
```

---

```
CSForm.getField("Entry1").setValue( rows.getValue("DependantName") );

// Close Rowset
rows.close();

// Close Connection
con.close();
}
```

### **close()**

Description:

Closes the CSRowSet object. Once the rowset is closed, the CSRowSet object and any derived column objects cannot be used.

Returns:

**Boolean**

Returns true on success and false on failure.

Example:

```
function CSForm_OnOpen()
{
/*
** Set Field Value equal to DependantName column's value
*/
var con = null; // DB Connection Object
var dLink = "PERF_TEST"; // Connect Agent Name
var table = "1040Dependants"; // Table Name
var sql = "SELECT * FROM \'" + table + "\'"; // SQL Statement to execute
var rows = null; // Rowset Object Variable

// Open New Connection
con = CSDB.newConnection( dLink );
if (!con) {
    CSForm.setStatusMsg( "Unable to open database connection!" );
    return;
}

// Open Rowset
rows = con.executeSQL( sql );
if (!rows) {
```

---

```
    CSForm.setStatusMsg( "Unable to open rowset!" );
    return;
}

// Set Value of Entry1 field to DependantName column's value
CSForm.getField("Entry1").setValue( rows.getValue("DependantName") );

// Close Rowset
rows.close();

// Close Connection
if (con.close())
    CSServer.log("Database Connection Closed.");
else
    CSServer.log("Error Closing Database Connection!");
}
```

---

### **nextRow()**

Description:

Move to next record in the rowset if any.

Returns:

**Boolean**

Returns true on success. Returns false if there are no more rows in the rowset or on failure.

Example:

```
function CSForm_OnOpen()
{
    var Columns = null; // Array for column list
    var con = null; // DB Connection Object
    var dLink = "PERF_TEST"; // Connect Agent Name
    var table = "1040Dependants"; // Table Name
    var sql = "SELECT * FROM \\" + table + "\\"; // SQL Statement to execute
    var rows = null; // Rowset Object Variable
    var dbList = CSForm.getField("dbList").getChoices();
    var cols = 0;

    // Open New Connection
```

---

```
con = CSDB.newConnection( dLink );
if (!con) {
    CSForm.setStatusMsg( "Unable to open database connection!" );
    return;
}

// Open Rowset
rows = con.executeSQL( sql );
if (!rows) {
    CSForm.setStatusMsg( "Unable to open rowset!" );
    return;
}

// Remember Column Count
cols = rows.getColumnCount();

// Add each row to the list field
while (rows.nextRow()) {
    var val = "";
    for (var i = 1; i <= cols; i++)
        val += rows.getValue(i) + ",";
    dbList.insertAt( -1, val, val );
}

// Close Rowset
rows.close();

// Close Connection
con.close();
}
```

---

#### **Number getRowIndex()**

Description:

Gets the current row number starting at 1.

---

Returns:

### Number

Returns 0 if there is no current row; the row position is either before the first row in which case nextRow() can be called to retrieve the first record, or after the last record. Returns -1 if an error occurred.

Example:

```
function CSForm_OnOpen()
{
    var Columns = null; // Array for column list
    var con = null; // DB Connection Object
    var dLink = "PERF_TEST"; // Connect Agent Name
    var table = "1040Dependants"; // Table Name
    var sql = "SELECT * FROM \\" + table + "\\"; // SQL Statement to execute
    var rows = null; // Rowset Object Variable
    var dbList = CSForm.getField("dbList").getChoices();
    var cols = 0;

    // Open New Connection
    con = CSDB.newConnection( dLink );
    if (!con) {
        CSForm.setStatusMsg( "Unable to open database connection!" );
        return;
    }

    // Open Rowset
    rows = con.executeSQL( sql );
    if (!rows) {
        CSForm.setStatusMsg( "Unable to open rowset!" );
        return;
    }

    // Remember Column Count
    cols = rows.getColumnCount();

    // Add each row to the list field
    while (rows.nextRow()) {
        var val = rows.getRowIndex() + "- ";
        for (var i = 1; i <= cols; i++)

```

---

```
    val += rows.getValue(i) + ",";
    dbList.insertAt( -1, i, val );
}

// Close Rowset
rows.close();

// Close Connection
con.close();
}
```

---

### **Boolean skip( Number n )**

Description:

Skips n rows. skip(1) is equivalent to calling nextRow().

Arguments:

**Number - n**

Number of rows to skip.

Returns:

**Boolean**

Returns true on success. Returns false on failure or if there are not n more rows in the rowset.

Example:

```
function CSForm_OnOpen()
{
    var Columns = null; // Array for column list
    var con = null; // DB Connection Object
    var dLink = "PERF_TEST"; // Connect Agent Name
    var table = "1040Dependants"; // Table Name
    var sql = "SELECT * FROM \\" + table + "\\"; // SQL Statement to execute
    var rows = null; // Rowset Object Variable
    var dbList = CSForm.getField("dbList").getChoices();
    var cols = 0;

    // Open New Connection
    con = CSDB.newConnection( dLink );
    if (!con) {
        CSForm.setStatusMsg( "Unable to open database connection!" );
    }
}
```

---

```
        return;
    }

    // Open Rowset
    rows = con.executeSQL( sql );
    if (!rows) {
        CSForm.setStatusMsg( "Unable to open rowset!" );
        return;
    }

    // Remember Column Count
    cols = rows.getColumnCount();

    // Add each row to the list field
    while (rows.skip(1)) {
        var val = rows.getRowIndex() + "- ";
        for (var i = 1; i <= cols; i++)
            val += rows.getValue(i) + ",";
        dbList.insertAt( -1, i, val );
    };

    // Close Rowset
    rows.close();

    // Close Connection
    con.close();
}
```

---

## CSColumn Object Specific Methods (Server Side)

The CSColumn object provides information about the columns returned in a CSRowSet. Use a CSRowSet to create CSColumn objects.

---

### *String* getName()

Description:

Gets the column name.

Returns:

**String**

Returns the column name as String.

Example:

```
function CSForm_OnOpen()
{
    var Columns = null; // Array for column list
    var con = null; // DB Connection Object
    var dLink = "PERF_TEST"; // Connect Agent Name
    var table = "1040Dependants"; // Table Name
    var sql = "SELECT * FROM \'" + table + "\'"; // SQL Statement to execute
    var rows = null; // Rowset Object Variable
    var dbList = CSForm.getField("dbList").getChoices();
    var cols = null; // Column Object Array
    var buff = ""; // String buffer

    // Open New Connection
    con = CSDB.newConnection( dLink );
    if (!con) {
        CSForm.setStatusMsg( "Unable to open database connection!" );
        return;
    }

    // Open Rowset
    rows = con.executeSQL( sql );
    if (!rows) {
        CSForm.setStatusMsg( "Unable to open rowset!" );
        return;
    }
```

---

```
// Retrieve Columns
cols = rows.getColumnList();
if (!cols) {
    CSForm.setStatusMsg( "Unable to retrieve columns!" );
    return;
}

// Add each column's info to the list field
for (var i = 0; i < rows.getColumnCount(); i++) {
    buff = "Name: " + cols[i].getName() +
        " - Type: " + cols[i].getType() +
        " - Size: " + cols[i].getSize() +
        " - Nullable: " + cols[i].isNullable();
    dbList.insertAt( i, i, buff );
}

// Close Connection
con.close();
}
```

---

### ***String getType()***

Description:

Gets the data type of the column.

Returns:

**String**

Returns the data type of the column as defined by the database. Typical values are "String", "Numeric", "Binary", "Date", "Time" or "TimeStamp", however some databases may return "varchar", "Number", "bin", "date", etc. These will vary with the database you use.

---

Example:

```
function CSForm_OnOpen()
{
    var Columns = null; // Array for column list
    var con = null; // DB Connection Object
    var dLink = "PERF_TEST"; // Connect Agent Name
    var table = "1040Dependants"; // Table Name
    var sql = "SELECT * FROM \\" + table + "\\"; // SQL Statement to execute
    var rows = null; // Rowset Object Variable
    var dbList = CSForm.getField("dbList").getChoices();
    var cols = null; // Column Object Array
    var buff = ""; // String buffer

    // Open New Connection
    con = CSDB.newConnection( dLink );
    if (!con) {
        CSForm.setStatusMsg( "Unable to open database connection!" );
        return;
    }

    // Open Rowset
    rows = con.executeSQL( sql );
    if (!rows) {
        CSForm.setStatusMsg( "Unable to open rowset!" );
        return;
    }

    // Retrieve Columns
    cols = rows.getColumnList();
    if (!cols) {
        CSForm.setStatusMsg( "Unable to retrieve columns!" );
        return;
    }

    // Add each column's info to the list field
    for (var i = 0; i < rows.getColumnCount(); i++) {
        buff = "Name: " + cols[i].getName() +
```

---

```
        " - Type: " + cols[i].getType() +
        " - Size: " + cols[i].getSize() +
        " - Nullable: " + cols[i].isNullable();
    dbList.insertAt( i, i, buff );
}

// Close Connection
con.close();
}
```

### **Number getSize()**

Description:

Gets the maximum number of characters allowed in the column. For numeric types this is the precision.

Returns:

**Number**

Returns the size of the column in characters or the precision if the column is a numeric data type.

Example:

```
function CSForm_OnOpen()
{
    var Columns = null; // Array for column list
    var con = null; // DB Connection Object
    var dLink = "PERF_TEST"; // Connect Agent Name
    var table = "1040Dependants"; // Table Name
    var sql = "SELECT * FROM \\" + table + "\\"; // SQL Statement to execute
    var rows = null; // Rowset Object Variable
    var dbList = CSForm.getField("dbList").getChoices();
    var cols = null; // Column Object Array
    var buff = ""; // String buffer

    // Open New Connection
    con = CSDB.newConnection( dLink );
    if (!con) {
        CSForm.setStatusMsg( "Unable to open database connection!" );
        return;
    }
```

---

```
// Open Rowset
rows = con.executeSQL( sql );
if (!rows) {
    CSForm.setStatusMsg( "Unable to open rowset!" );
    return;
}

// Retrieve Columns
cols = rows.getColumnList();
if (!cols) {
    CSForm.setStatusMsg( "Unable to retrieve columns!" );
    return;
}

// Add each column's info to the list field
for (var i = 0; i < rows.getColumnCount(); i++) {
    buff = "Name: " + cols[i].getName() +
        " - Type: " + cols[i].getType() +
        " - Size: " + cols[i].getSize() +
        " - Nullable: " + cols[i].isNullable();
    dbList.insertAt( i, i, buff );
}

// Close Connection
con.close();
}
```

---

#### ***Boolean isNullable()***

Description:

Indicates whether the column can be left null or not.

Returns:

**Boolean**

Returns true if the column can contain null values or false if a value must be set.

---

Example:

```
function CSForm_OnOpen()
{
    var Columns = null; // Array for column list
    var con = null; // DB Connection Object
    var dLink = "PERF_TEST"; // Connect Agent Name
    var table = "1040Dependants"; // Table Name
    var sql = "SELECT * FROM \\" + table + "\\"; // SQL Statement to execute
    var rows = null; // Rowset Object Variable
    var dbList = CSForm.getField("dbList").getChoices();
    var cols = null; // Column Object Array
    var buff = ""; // String buffer

    // Open New Connection
    con = CSDB.newConnection( dLink );
    if (!con) {
        CSForm.setStatusMsg( "Unable to open database connection!" );
        return;
    }

    // Open Rowset
    rows = con.executeSQL( sql );
    if (!rows) {
        CSForm.setStatusMsg( "Unable to open rowset!" );
        return;
    }

    // Retrieve Columns
    cols = rows.getColumnList();
    if (!cols) {
        CSForm.setStatusMsg( "Unable to retrieve columns!" );
        return;
    }

    // Add each column's info to the list field
    for (var i = 0; i < rows.getColumnCount(); i++) {
        buff = "Name: " + cols[i].getName() +
```

---

```
    " - Type: " + cols[i].getType() +
    " - Size: " + cols[i].getSize() +
    " - Nullable: " + cols[i].isNullable();
    dbList.insertAt( i, i, buff );
}

// Close Connection
con.close();
}
```

---

## Common Script Entry Points (Server Side)

In some cases, system administrators may need to run script regardless of which form is currently being filled by a user. Common Script Entry Points allow this script to be written once and then applied to ALL forms on the server.

Common (Global) scripts like all other scripts are event driven, they will run every time a specific event occurs. The events that are provided for Common scripts are based on those provided at the form level (CSForm\_OnOpen, CSForm\_OnSubmit, CSForm\_OnRoutePage, CSForm\_OnRoute, CSForm\_OnExport, and CSForm\_OnValidateLookup). There are two common events for each form level event; one that occurs just before the form event and one that occurs just after it. Common form level events are:

```
CSForm_OnOpenBefore  
CSForm_OnOpenAfter  
CSForm_OnSubmitBefore  
CSForm_OnSubmitAfter  
CSForm_OnRoutePageBefore  
CSForm_OnRoutePageAfter  
CSForm_OnRouteBefore  
CSForm_OnRouteAfter  
CSForm_OnExportBefore  
CSForm_OnExportAfter  
CSForm_OnValidateLookupBefore  
CSForm_OnValidateLookupAfter
```

Additionally, two server level common events have been provided:

```
OnServerInit  
OnServerExit
```

OnServerInit occurs just after the LiquidOffice Server starts. OnServerExit occurs just before the server terminates. The actual script is saved in a file called commands.js on the LiquidOffice Server. This file is located in the folder serverscripts\command found under the LiquidOffice folder (Default installation: C:\Program Files\Cardiff\LiquidOffice\server\serverscripts\command\command.js). By default, a sample commands.js file will be created on the server containing prototypes of all common entry points. Note: The entire contents of the sample commands.js file has been commented out. In order for the script to run, uncomment the entry points you wish to use.

When LiquidOffice Server starts, the commands.js script is loaded into memory so that it can be run when needed. There will be times either during script development or updates when the system administrator will need to reload the commands.js script file rather than stopping and restarting the server. This is done in the Server Administration area of the LiquidOffice Server Web Desktop. There is a button labeled 'Reload Plug-ins' in the Options section of the Server Administration area. Clicking on this button will cause the commands.js script to be reloaded along with any Custom Export Scripts and all other plug-ins.

---

## Form Level Common Events

---

### **CSForm\_OnOpenBefore()**

Description:

Occurs just before the CSForm\_OnOpen event for each form would occur as it is opened by the end user.

Example:

```
function CSForm_OnOpenBefore()
{
    CSServer.log("Form: " + CSForm.getTitle() + " about to open...");
}
```

---

### **CSForm\_OnOpenAfter()**

Description:

Occurs just after the CSForm\_OnOpen event for each form would occur as it is opened by the end user.

Example:

```
function CSForm_OnOpenAfter()
{
    CSServer.log("Form: " + CSForm.getTitle() + " now open.");
}
```

---

### **CSForm\_OnSubmitBefore()**

Description:

Occurs just before the CSForm\_OnSubmit event for each form would occur before it is submitted by the end user.

Example:

```
function CSForm_OnSubmitBefore()
{
    CSServer.log("Form Submission About to Take Place\r\n" +
        "FormID: " + CSForm.getID() + "\r\n" +
        "UserID: " + CSServer.getCurrentUser().getUserID());
}
```

---

### **CSForm\_OnSubmitAfter()**

Description:

Occurs just after the CSForm\_OnSubmit event for each form would occur before it is submitted by the end user.

Example:

```
function CSForm_OnSubmitAfter()
{
    CSServer.log("Form Submission Completed\r\n" +
        "FormID: " + CSForm.getID() + "\r\n" +
        "UserID: " + CSServer.getCurrentUser().getUserID() );
}
```

---

### **CSForm\_OnRoutePageBefore()**

Description:

Occurs just before the CSForm\_OnRoutePage event for each form would occur before the Routing Page is displayed to the end user.

Example:

```
function CSForm_OnRoutePageBefore()
{
    CSServer.log("Before Route Page Event" + "\r\n" +
        "FormID: " + CSForm.getID() + "\r\n" +
        "RouteTo: " + CSRoute.getRouteTo());

    CSRoute.setRouteTo( CSServer.getCurrentUser().getUserID() );
}
```

---

### **CSForm\_OnRoutePageAfter()**

Description:

Occurs just after the CSForm\_OnRoutePage event for each form would occur before the Routing Page is displayed to the end user.

Example:

```
function CSForm_OnRoutePageAfter()
{
    CSServer.log("After Route Page Event" + "\r\n" +
        "FormID: " + CSForm.getID() + "\r\n" +
        "RouteTo: " + CSRoute.getRouteTo());
}
```

---

### **CSForm\_OnRouteBefore()**

Description:

Occurs just before the CSForm\_OnRoute event for each form would occur just before the form is routed.

Example:

```
function CSForm_OnRouteBefore()
{
    CSServer.log("Before Route Event" + "\r\n" +
        "FormID: " + CSForm.getID() + "\r\n" +
        "RouteTo: " + CSRoute.getRouteTo());
}
```

---

### **CSForm\_OnRouteAfter()**

Description:

Occurs just after the CSForm\_OnRoute event for each form would occur just before the form is routed.

Example:

```
function CSForm_OnRouteAfter()
{
    CSServer.log("After Route Event" + "\r\n" +
        "FormID: " + CSForm.getID() + "\r\n" +
        "RouteTo: " + CSRoute.getRouteTo());
}
```

---

### **CSForm\_OnExportBefore()**

Description:

Occurs just before the CSForm\_OnExport event for each form would occur when a form is finalized and ready for data to be committed to a database.

Example:

```
function CSForm_OnExportBefore()
{
    CSServer.log( "Form Export Ready" + "\r\n" +
        "FormID: " + CSForm.getID() + "\r\n" +
        "Form: " + CSForm.getTitle());
}
```

---

### **CSForm\_OnExportAfter()**

Description:

Occurs just after the CSForm\_OnExport event for each form would occur when a form is finalized and ready for data to be committed to a database.

Example:

```
function CSForm_OnExportAfter()
{
    CSServer.log( "Form Export Complete" + "\r\n" +
        "FormID: " + CSForm.getID() + "\r\n" +
        "Form: " + CSForm.getTitle() );
}
```

---

### **CSForm\_OnValidateLookupBefore()**

Description:

Occurs just before the CSForm\_OnValidateLookup event for each form would occur after a field validation or database lookup is completed.

Example:

```
function CSForm_OnValidateLookupBefore()
{
    CSServer.log("Before Validate Event" + "\r\n" +
        "FormID: " + CSForm.getID() + "\r\n" +
        "Lookup Trigger: " + CSEvent.getTarget() );
}
```

---

### **CSForm\_OnValidateLookupAfter()**

Description:

Occurs just after the CSForm\_OnValidateLookup event for each form would occur after a field validation or database lookup is completed.

Example:

```
function CSForm_OnValidateLookupAfter()
{
    CSServer.log("After Validate Event" + "\r\n" +
        "FormID: " + CSForm.getID() + "\r\n" +
        "Lookup Trigger: " + CSEvent.getTarget() );
}
```

---

### **OnServerInit()**

Description:

Occurs just after the server is started.

Example:

```
function OnServerInit()
{
    CSServer.log("LiquidOffice Server Started" + "\r\n" +
        "Version: " + CSServer.getVersion() );
}
```

---

### **OnServerExit()**

Description:

Occurs just before the server is terminated.

Example:

```
function OnServerExit()
{
    CSServer.log("LiquidOffice Server Stopping... Goodnight.");
}
```

---

## Response Pages

When designing forms in LiquidOffice Form Designer, you can write JavaScript to display a response page either when the user submits or routes the form. Use the Client-side OnSubmit event or Server-side OnSubmit or OnRoute event to set a response page for a particular form. The response page can be set as one of the following items:

- URL Response Page
- Custom URL Response Page
- LiquidOffice Form Response Page

### URL Response Page

This version of LiquidOffice Form Designer provides a scripting method for implementing response pages. You can use script to set a URL when the user submits or routes the form.

#### To add a URL response page using script, follow these steps:

1. Open the form to which you want to add the script.
2. Launch the **LiquidOffice JavaScript Editor** from the **Form** menu.
3. Click **Client > CSForm** [in the Navigation pane].
4. Double-click **OnSubmit** or **OnRoute** (server only) to add the event handler depending on when you want the custom response page to appear.

NOTE: Use the OnRoute to set custom response pages. To do so, use the JavaScript Editor and click on **Server>CSForm>OnSubmit** or **OnRoute** and add the script.

5. Write JavaScript using the **CSForm.setResponseURL** method to set the Custom Response Page URL when the form is submitted.

**Example:** Calls the specified URL <http://www.Cardiff.com> when the form is submitted.

```
function CSForm_OnSubmit() {  
    CSForm.setResponseURL("http://www.Cardiff.com" );  
    return true;  
}
```

---

## Custom URL Response Page

When adding a response page, you can also customize the page by passing a parameter to the page using JavaScript. You can pass the value of a form field to the response page using a parameter. If you use the Name field, for example, the response page could read “Thank You John!”.

To create a custom response page using script, follow these steps:

1. Open the form to which you want to add the script.
2. Launch the **LiquidOffice JavaScript Editor** from the **Form** menu.
3. Click **Client > CSForm** [in the Navigation pane].

**NOTE:** You can also use the server script methods **OnSubmit** or **OnRoute** to set custom response page. To do so, use the JavaScript Editor and click on **Server>CSForm>OnSubmit** or **OnRoute** and add the script.

4. Double-click **OnSubmit** or **OnRoute** to add the event handler.
5. Write JavaScript to pass as a parameter any value that was captured earlier on the form.
6. Upload the response page to a server i.e. <http://myserver/mydir/>.

**Example:** *Calls a URL and passes a parameter to the response page.*

```
function CSForm_OnSubmit() {  
    var aName = CSForm.getField("Name").getValue();  
    CSForm.setResponseURL( "http://myserver/mydir/myResponse.html?param1=" + aName );  
    return true;  
}
```

---

## Example Custom Response Page

This example displays a Custom Response Page that uses the captured value of a particular field.

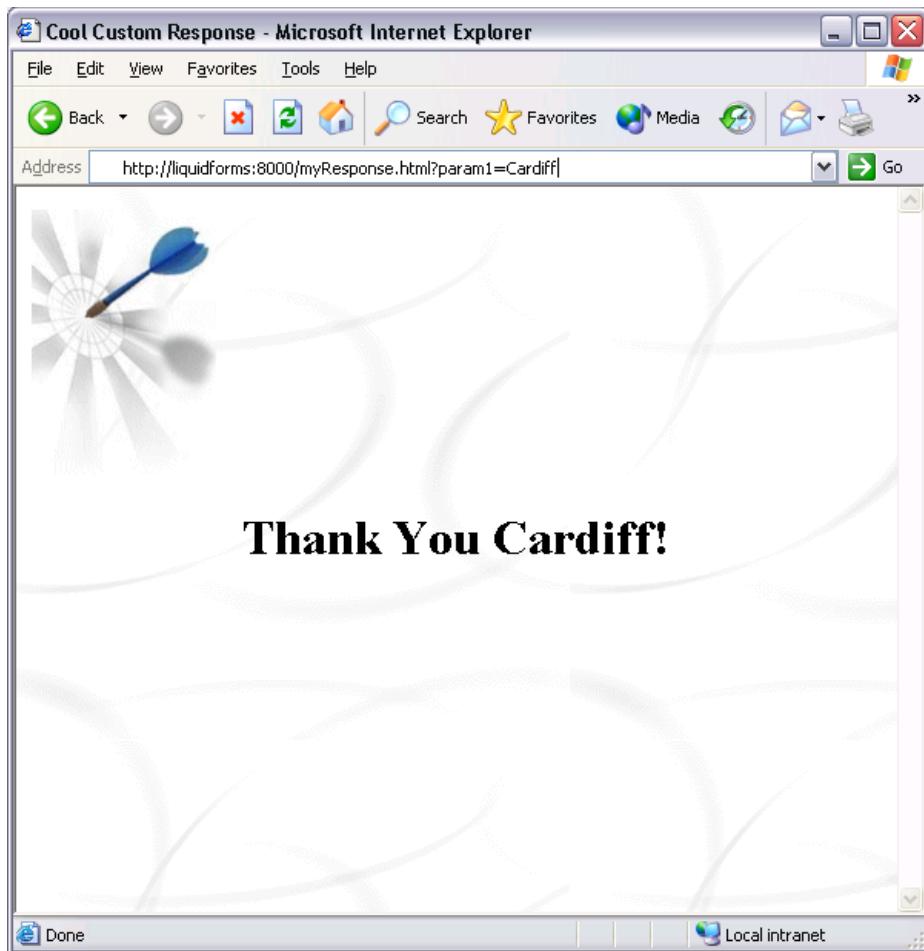
```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.0 Transitional//EN">

<html>
<head>
<title>Cool Custom Response</title>
<script language="JavaScript" type="text/javascript">
<!--
var s = new String(self.location.href);
var i = s.indexOf("?param1=");
var v = "";
if (i>0) v= s.substr(i+8);

//-->
</script>
</head>
<body background="main_bg.jpg">
<blockquote>
<script language="JavaScript" type="text/javascript">
document.write("<h1>Thank You "+v+"!</h1>");
</script>
</blockquote>
</body>
</html>
```

---

## Example Response Page



---

## LiquidOffice Form Response Page

In addition to customizing response pages by passing parameter through a URL, you can also write script to open a LiquidOffice Form for both OnSubmit and OnRoute CSForm events. For example, if the user needs to fill out a series of forms, you might want to launch a particular form as a response page so that the user can complete the next form in the series.

### To set a LiquidOffice Form as a Custom Response Page, follow these steps:

1. Open the form to which you want to add the script.
2. Launch the **LiquidOffice JavaScript Editor** from the **Form** menu.
3. Click **Client > CSForm** [in the Navigation pane].

NOTE: You can also use the server script methods OnSubmit or OnRoute to set custom response page. To do so, use the JavaScript Editor and click on **Server>CSForm>OnSubmit** or **OnRoute** and add the script.

4. Double-click **OnSubmit** or **OnRoute** (server only) to add the event handler depending on when you want the custom response page to appear.
5. Write JavaScript using the CSForm.setResponseURL method to set the Custom Response Page URL when the form is submitted.

**Example:** Calls a LiquidOffice Form URL and passes a parameter to the response page.

```
function CSForm_OnSubmit() {  
    var aName = CSForm.getField("Name").getValue();  
    CSForm.setResponseURL( "http://myLiquidOffice/Public/MyForm?Name=" + aName );  
    return true;  
}
```

## Server Object/Method Availability

**Event Availability Table -- Events called for these actions**

	EVENT							
		Other Events	CSForm_OnValidate Lookup*	CSForm_OnSubmit*	CSForm_OnRoute Page*	CSForm_OnRoute*	CSForm_OnExport*	Export Script*
O B J E C T / M E T H O D	<b>Open</b>	CSForm_OnOpen*						
	<b>Submit</b>		Yes	Yes	Yes	Yes	If Final	If Final
	<b>Approve</b>		Yes	Yes	Yes	Yes	If Route Complete	If Route Complete
	<b>Transfer</b>		Yes	Yes	Yes	Yes		
	<b>Reject</b>			Yes	Yes (not multiple)	Yes		
	<b>Redirect</b>				Yes	Yes		
	<b>Field tab out</b>		Lookup or Validation					
	<b>Button click</b>		Lookup					
	<b>Server Startup**</b>	OnServerInit						
	<b>Server Shutdown**</b>	OnServerExit						

\* Includes Before and After methods such as CSForm\_OnOpenBefore and CSForm\_OnOpenAfter.

\*\* Startup and Shutdown occur on Reload Plugins also.

## Object/Method Availability Table

EVENT							
		CSForm_	CSForm_	CSForm_	CSForm_	Export	On
		OnOpen*	OnValidate	OnSubmit*	OnRoute*	Script*	Server
<b>CSEvent</b>	Yes	Yes	Yes	Yes	Yes	Yes	Exit
getTarget	Form	Field	Form	Form	Form	Form	
getAction	Yes	Yes	Yes	Yes	Yes	Yes	
<b>CServer</b>	Yes	Yes	Yes	Yes	Yes	Yes	Yes
<b>CSForm</b>	Yes	Yes	Yes	Submit, Approve, Transfer	Submit, Approve, Transfer		
setFocus	Yes	Yes	Yes	Submit, Approve, Transfer	Submit, Approve, Transfer	Yes	
getPukType	Yes	Yes	Yes	Submit, Approve, Transfer	Submit, Approve, Transfer	Yes	
getTitle	Yes	Yes	Yes	Submit, Approve, Transfer	Submit, Approve, Transfer	Yes	
<b>J</b>	Yes	Yes	Yes	Submit, Approve, Transfer	Submit, Approve, Transfer	Yes	
<b>E</b>	Yes	Yes	Yes	Submit, Approve, Transfer	Submit, Approve, Transfer	Yes	
<b>C</b>	Yes	Yes	Yes	Submit, Approve, Transfer	Submit, Approve, Transfer	Yes	
<b>T</b>	Yes	Yes	Yes	Submit, Approve, Transfer	Submit, Approve, Transfer	Yes	
<b>Y</b>							
<b>M</b>	Yes	Yes	Yes	Yes	Yes	Yes	
<b>E</b>	Yes	Yes	Yes	Yes	Yes	Yes	
<b>T</b>	Yes	Yes	Yes	Yes	Yes	Yes	
<b>H</b>	Yes	Yes	Yes	Yes	Yes	Yes	
<b>O</b>	Yes	Yes	Yes	Yes	Yes	Yes	
<b>D</b>	Yes	Yes	Yes	Yes	Yes	Yes	
getNumNumberOffFields	Yes	Yes	Yes	Submit, Approve, Transfer	Submit, Approve, Transfer	Yes	
getField	Yes	Yes	Yes	Submit, Approve, Transfer	Submit, Approve, Transfer	Yes	
getDocument				Yes	Yes	Yes	
mergePDF				Yes	Yes	Yes	
getAttachments						Yes	
getAttachmentURLs						Yes	
getLocation	Yes	Yes	Yes	Yes	Yes	Yes	
getFinalized	Yes	Yes	Yes	Yes	Yes	Yes	
getLanguageCode	Yes	Yes	Yes	Yes	Yes	Yes	
getResponseURL				Submit	Submit		
setResponseURL				Yes	Submit		

**Event Availability Table -- Events called for these actions**

EVENT										
		CSForm_OnOpen*	CSForm_OnValidate Lookup*	CSForm_OnSubmit*	CSForm_OnRoute Page*	CSForm_OnRoute*	CSForm_OnExport*	Export Script*	On Server Init	On Server Exit
<b>O B J E C T / M E T H O D</b>	<b>CSRoute</b>	Yes	Yes	Yes	Yes	Yes	Yes			
	getRouteTo				Yes	Yes				
	setRouteTo				Submit, Approve, Transfer, Redirect	Submit, Approve, Transfer, Redirect				
	getCCTo				Yes	Yes	Yes			
	addCCTo				Yes	Yes				
	getNotes				Yes	Yes				
	setNotes				Yes	Yes				
	getStatus		Yes	Yes	Yes	Yes				
	getLevel	Yes	Yes	Yes	Yes	Yes	Yes			
	getRoutingComplete				Submit, Approve	Submit, Approve				
	setRoutingComplete				Submit, Approve	Submit, Approve				
	setShowPage				Yes					
	setEnableRouteToCtrl				Submit, Approve, Transfer, Redirect					
	setEnableCCToCtrl				Yes					
	<b>CSUser</b>	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	
	<b>CSFile</b>	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	
	<b>CSMail</b>	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	
	<b>CSDB</b>	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes

Other objects such as CSField, CSChoice, etc. are available in all methods. However since they are always derived from one of the above listed objects, their availability really depends on the parent object.